الگوریتم‌های فشرده‌سازی بدون تلف (Lossless)

خلاصهٔ فصل

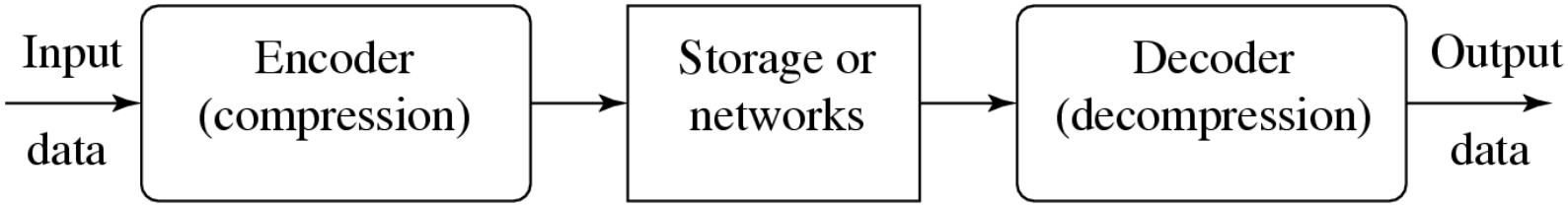
- تئوری اطلاعات
- مفاهیم کدگذاری
- کدگذاری باطول متغیر
- کدگذاری مبتنی بر دیکشنری و حسابی
- فشرده‌سازی بدون اتلاف تصاویر

هدف

در این بخش با مقدماتی از تئوری اطلاعات و مفهوم آنتروپی آشنا می‌شویم.

مقدمات تئوری اطلاعات

- **فشرده‌سازی (Compression)**: به فرایند رمز کردن که منجر به کاهش موثر تعداد کل بیت‌های لازم برای ارائه اطلاعات مشخصی می‌شود ، فشرده‌سازی گفته می‌شود.



شمای کلی از فشرده‌سازی

مقدمات تئوری اطلاعات

- اگر فرایند فشرده‌سازی و بازگرداندن به حجم اولیه ، بدون از دست رفتن اطلاعات صورت گیرد ، فشرده سازی بدون تلف (Lossless) است و در غیر این صورت با تلف (Lossy) است.

- **ضریب فشرده‌سازی:**

$$\text{ضریب فشرده سازی} = \frac{B_0}{B_1}$$

B0 - تعداد بیت‌ها قبل از فشرده‌سازی

B1 – تعداد بیت‌ها بعد از فشرده‌سازی

سوال

- اگر تعداد بیت‌ها پس از فشرده‌سازی نصف بشه میزان نرخ فشرده‌سازی چقدر خواهد بود ؟
  - 0/5
  - 2
  - 1/5
  - 0/66

مقدمات تئوری اطلاعات

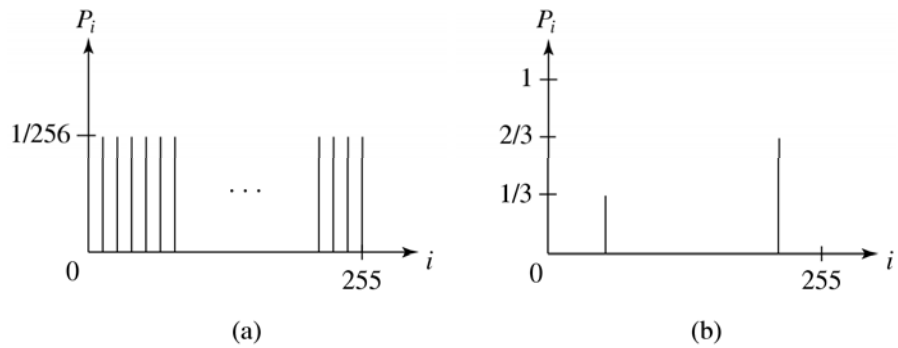
- آنتروپی η از یک منبع اطلاعاتی $S=\{s_1, s_2, \ldots, s_n\}$ برابر است با:

$$\eta = H(S) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i}$$

$$= -\sum_{i=1}^{n} p_i \log_2 p_i$$

p_i - احتمال رخداد s_i در S

مشخص‌کنندهٔ میزان اطلاعات موجود در s_i موجود است ، بنابراین می‌تواند تعیین کنندهٔ تعداد بیت‌های مورد نیاز برای کد کردن s_i باشد. $\log_2 \frac{1}{p_i}$



هیستوگرام سطح خاکستری دو تصویر

- (a) هیستوگرام یک تصویر با شدت‌های توزیع سطح خاکستری یک‌نواخت و مساوی را نشان می‌دهد ، به ازای هر i داریم:

$p_i = 1/256$ بنابراین آنتروپی این تصویر برابر است با:

$$\sum_{i=1}^{256} \frac{1}{256} \log_2 256 = 8$$

$\log_2 256 = 8$

- (b) هیستوگرام یک تصویر با دو مقدار ممکن را نشان میدهد. آنتروپی آن 0.92 است.

$$\frac{1}{3}\log_2 3 + \frac{2}{3}\log_2 \frac{3}{2}$$

آنتروپی و طول کد

- آنتروپی η یک مجموع وزنی از مؤلفه‌های $\log_2 \frac{1}{p_i}$ است ، بنابراین نشان‌دهندهٔ میانگین مقدار اطلاعات موجود در هر منبع S است.
- آنتروپی η حد پایین برای میانگین تعداد بیت‌ها برای کدکردن هر مجموعه سمبل S را مشخص می‌کند.

$$\eta \leq \bar{l}$$

- $\bar{l}$ میانگین طول (بر حسب بیت) کدهای تولید شده توسط کدکننده است.

سوال

- آنتروپی یک منبع قطعی (deterministic) چقدر است ؟

$$\eta = H(S) = \sum_{i=1}^{n} 1 \times \log_2 \frac{1}{1} = 0$$

تمرین

- اگر دو سمبل A و B در سیستم موجود باشند ، و احتمال حضور هر یک به این ترتیب تعریف شود: P(A)=x و P(B)=1-x بطوری‌که 0<=x<=1. به ازای چه مقداری از x آنتروپی ماکزیمم می‌شود ؟ مسئله را به‌صورت تحلیلی حل کنید.

هدف

در این بخش روش کدگذاری Run-Lenght ارئه خواهیم نمود.

کدینگ Run-Length (RLC)

- **منبع بدون حافظه:** یک منبع اطلاعات که به صورت مستقل توزیع شده است و مقادیر سمبل‌های متوالی به مقادیر سمبل‌های قبلی بستگی ندارد.
- کدگذاری RLC از نحوهٔ چینش اطلاعات در منبع استفاده می‌کند.

- **منطق اصلی RLC:** اگر منبع اطلاعات طوری باشد که نمادها به تشکیل گروه‌های پیوسته تمایل داشته باشند ، می‌توان این نمادها و طول گروه هر یک را برای کدکردن استفاده نمود.

RLC

- مثال: فرض کنید تصویری سیاه سفید داریم که به ترتیب زیر نمایش داده شده است. (W = سفید و B = سیاه)

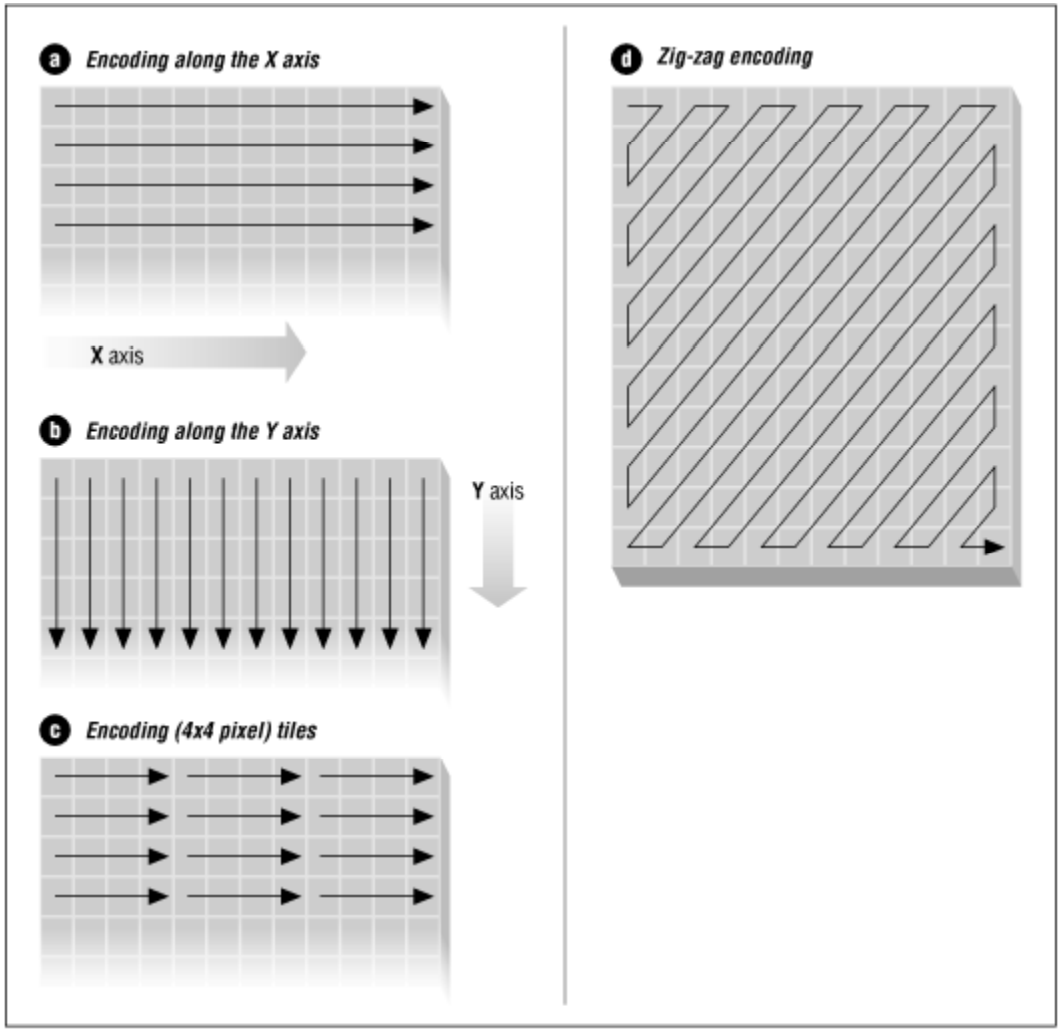
WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWWWBW
WWWWWWWWWWWWW

- پس از کدگذاری به این رشته می‌رسیم:

12W1B12W3B24W1B14W

- کاربردها:
  - بخشی از الگوریتم JPEG
  - Truevision TGA ، PackBits ، PCX و ILBM
  - در دستگاه‌های فکس
  - کاربردهای متفاوت بدون تلف

RLC


a Encoding along the X axis
X axis
b Encoding along the Y axis
Y axis
c Encoding (4x4 pixel) tiles
d Zig-zag encoding

سوال

- اگر تصویری باینری داشته باشیم که مقادیر پیکسل آن در I نشان داده شده باشد ، آن را با الگوریتم RLC و با پیمایش Zig-zag کد کنید.

$$I = \begin{matrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{matrix}$$

$$(1,1)(1,0)(4,1)(4,0)(1,1)(4,0)(1,1)(2,0)(1,1)(2,0)(2,1)(13,0)$$

هدف

در این بخش روش کدگذاری با طول متغیر را ارائه خواهیم نمود.

کدگذاری با طول متغیر

- فرض کنید که فایلی با 100،000کاراکتر را می خواهیم ذخیره کنیم و در این فایل فقط از 6 نوع کاراکتر مختلف استفاده شده است و میزان فراوانی کاراکترها در فایل به صورت زیر است:

	a	b	c	d	e	f
Frequency in ’000s	45	13	12	16	9	5

- یک کد دودویی هر کاراکتر را به صورت یک رشته دودویی کد می کند.
- ما به دنبال یافتن یک کد دودویی هستیم که فایل را با کمترین تعداد بیت کد کند(فشرده سازی تا حد ممکن)

کدگذاری با طول متغیر

- در یک کدینگ با طول ثابت طول رشته کد ساخته شده برای تمام کاراکترها یکسان است.
- در کدینگ با طول متغیر رشته کدها می توانند طول متغیر داشته باشند.
- در زیر کدینگ با طول ثابت و متغیر را برای مثال ذکر شده مشاهده می کنید

	a	b	c	d	e	f
Freq in ’000s	45	13	12	16	9	5
a fixed-length	000	001	010	011	100	101
a variable-length	0	101	100	111	1101	1100

- دقت کنید که در این مثال برای هر کاراکتر در کدینگ ثابت 3 بیت نیاز است. پس در کدینگ ثابت به 300،000 بیت برای ذخیره فایل نیاز است اما در کدینگ متغیر به 224،000 بیت نیاز است. چرا ؟
- با استفاده از کدینگ متغیر می توان در فضای ذخیره سازی صرفه جویی کرد.

کدینگ Variable-Lenght (VLC)

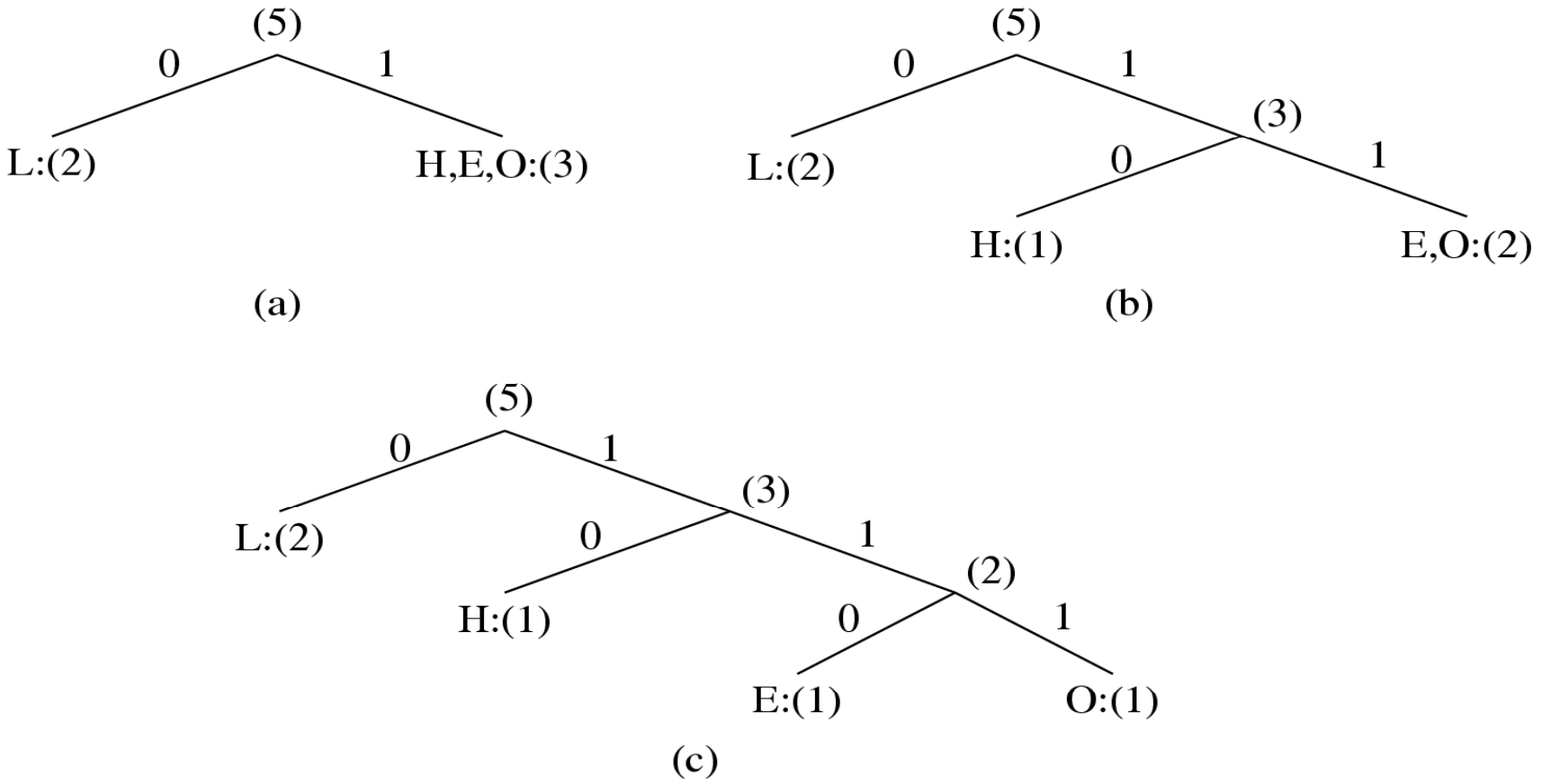
- **الگوریتم شانون-فانو (روش بالا به پایین):**

1. نمادها را بر اساس شمار فراوانی رخدادشان مرتب کنید.
2. نمادها را به صورت بازگشتی به دو قسمت تقسیم کنید ، هر کدام با تعداد شماره‌های تقریبا برابر ، تا جایی‌که هر قسمت فقط یک نماد داشته باشد.

- مثال: کدینگ کلمهٔ HELLO

نماد	H	E	L	O
تعداد	1	1	2	1

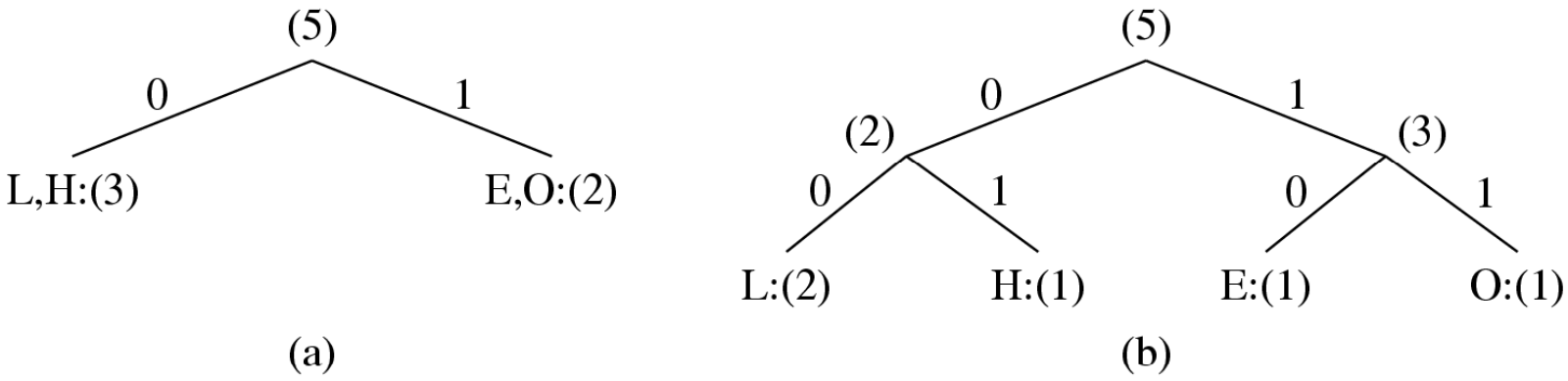
شمار فراوانی نمادها در HELLO



درخت کدینگ برای HELLO توسط شانون-فانو

نتایج اجرای شانون-فانو بر روی HELLO

Symbol	Count	$\log_2 \frac{1}{p_i}$	Code	# of bits used
L	2	1.32	0	2
H	1	2.32	10	2
E	1	2.32	110	3
O	1	2.32	111	3
			TOTAL # of bits:	10



درخت کدینگ دیگری توسط شانون-فانو برای HELLO

نتایج دیگر از اجرای شانون-فانو بر روی HELLO

Symbol	Count	$\log_2 \frac{1}{p_i}$	Code	# of bits used
L	2	1.32	00	4
H	1	2.32	01	2
E	1	2.32	10	2
O	1	2.32	11	2
			TOTAL # of bits:	10

- **الگوریتم کدینگ هافمن** (Huffman) روند پایین به بالا:

1. تمام نمادها را بر اساس شمار فراوانی آن‌ها در یک لیست مرتب کنید.
2. تا زمانی‌که فقط یک نماد باقی بماند ، ادامه دهید:

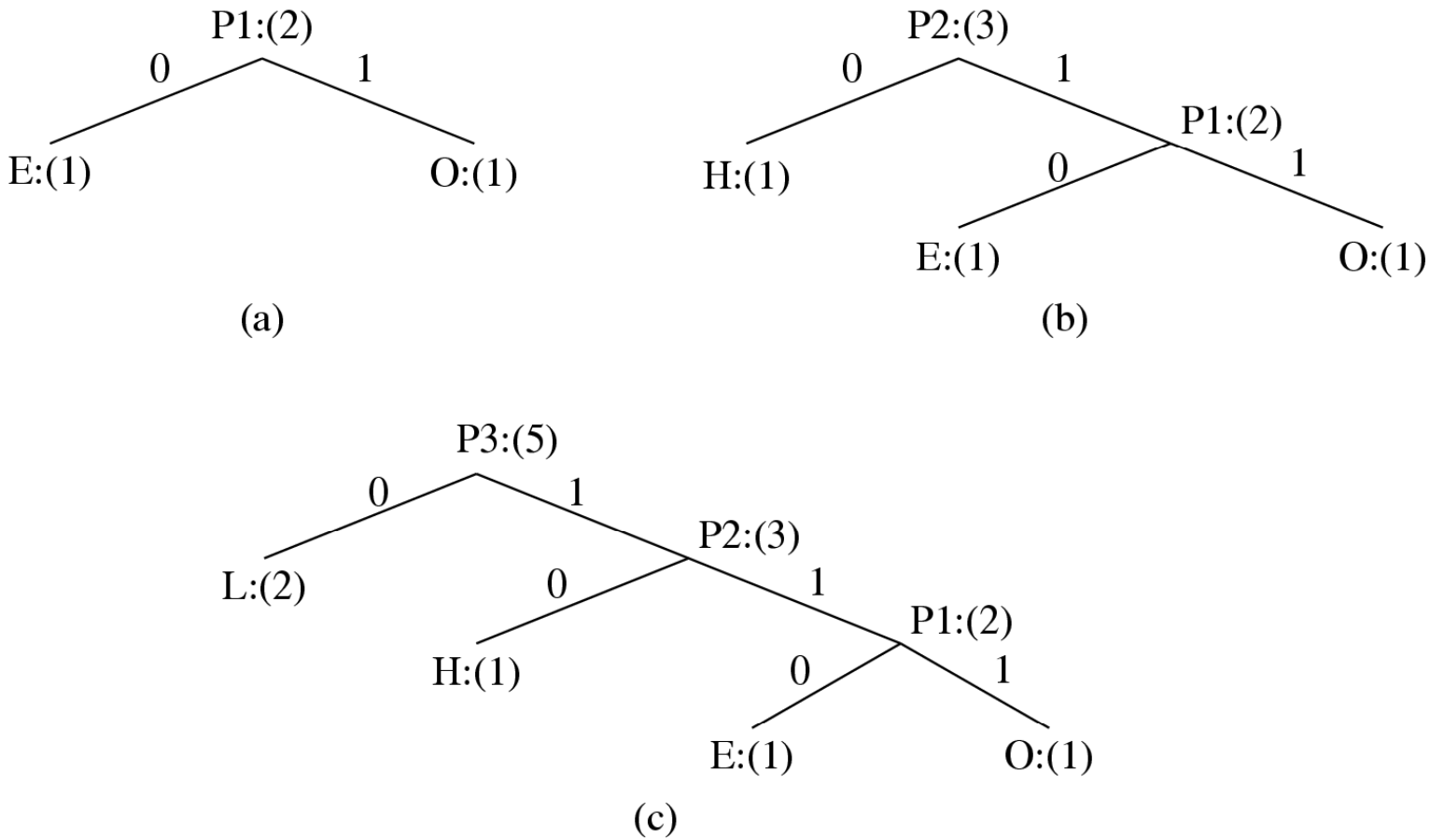
(1) دو نمادی که دارای کمترین فراوانی هستند را از لیست بردارید. یک زیر درخت هافمن ایجاد کنید که این دو نماد را به عنوان گره فرزند قرار داده و یک گره پدر ایجاد کنید.

(2) جمع تعداد فرزندان را به نود پدر اختصاص داده و نود پدر را در لیست به عنوان یک عنصر جدید درج کنید.

(3) فرزندان را از لیست حذف کنید.

(4) به مرحله 1 بروید

VLC



درخت کدینگ برای HELLO با استفاده از الگوریتم هافمن

VLC

- نمادهای جدید p1 ، p2 ، p3 برای اشاره به نودهای پدر در درخت کدینگ هافمن تولید شده‌اند. محتویات موجود در لیست ، در زیر مشخص شده است:

شروع:	L	H	E	O
بعد از تکرار (a):	L	P1	H	
بعد از تکرار (b):	L	P2		
بعد از تکرار (c):	P3			

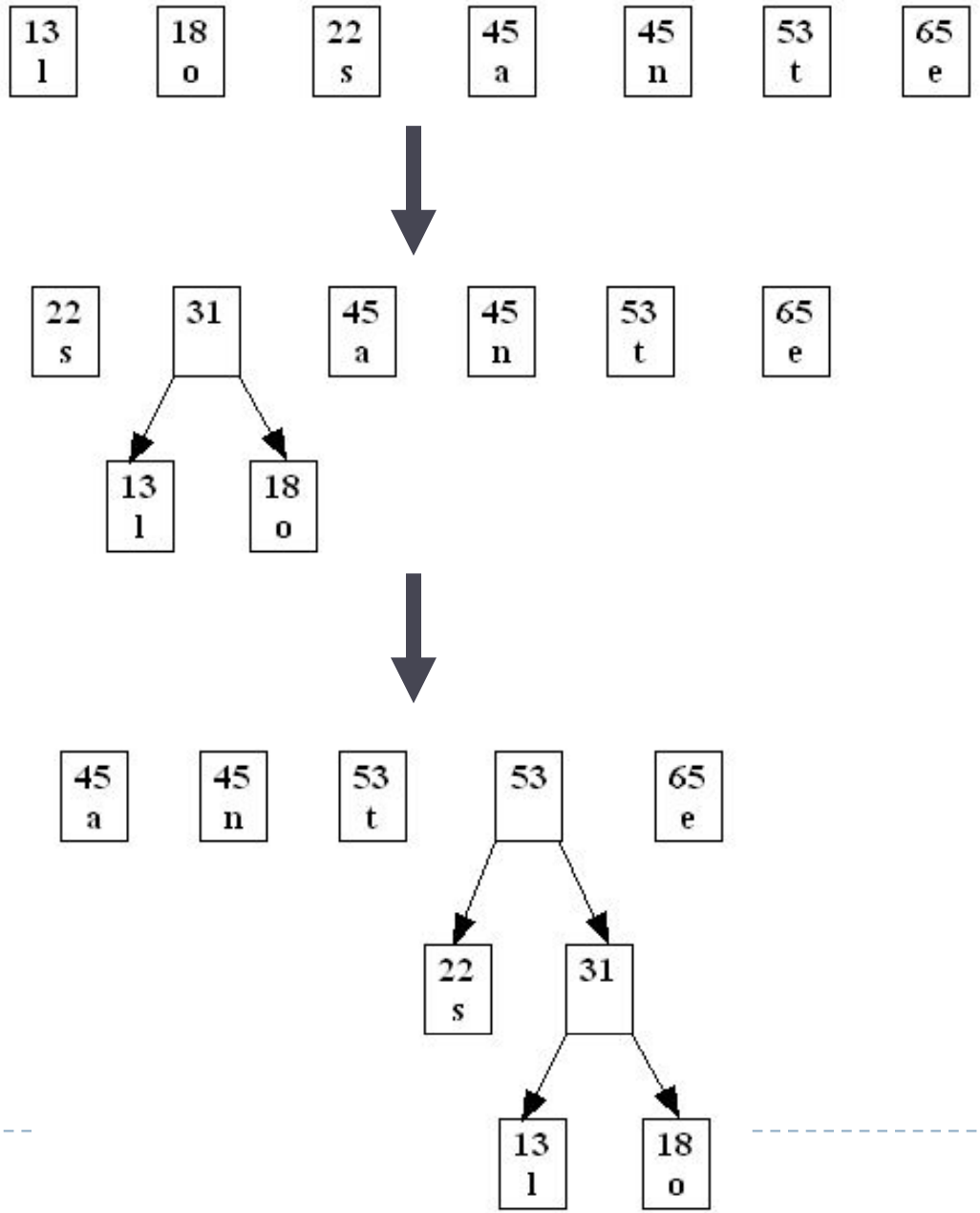
مثال کد هافمن

Character	a	e	l	n	o	s	t
Frequency	45	65	13	45	18	22	53

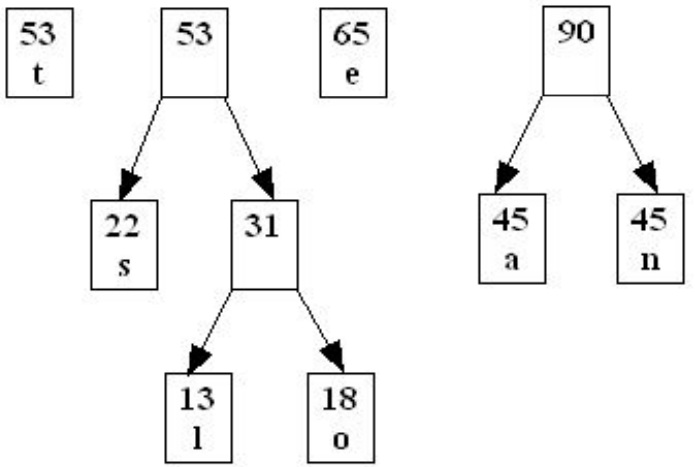
داده های بالا قرار است از طریق شبکه منتقل شوند.

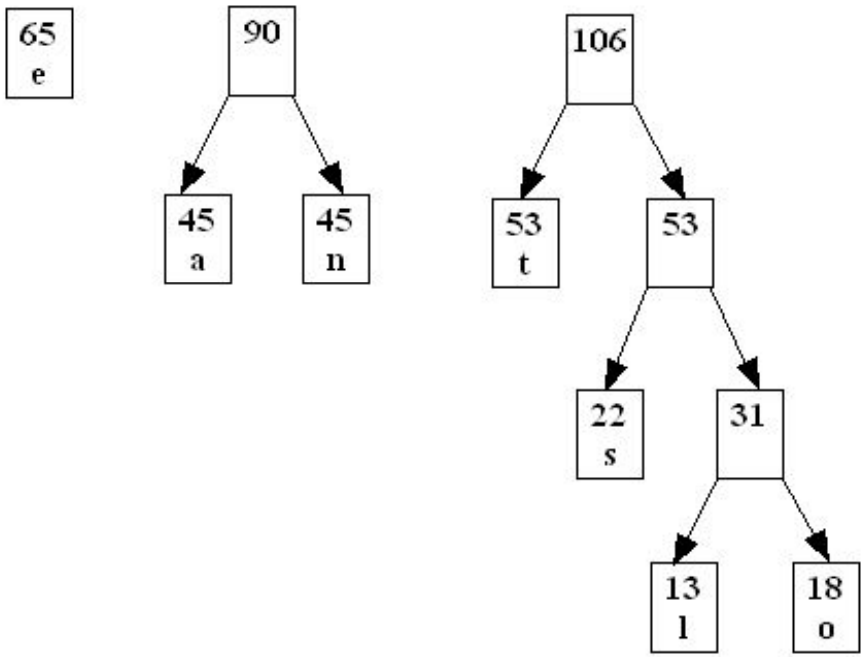
- کد هافمن را از طریق درخت هافمن برای آنها به دست آورید و درصد فشرده سازی را حساب کنید؟

مثال کد هافمن

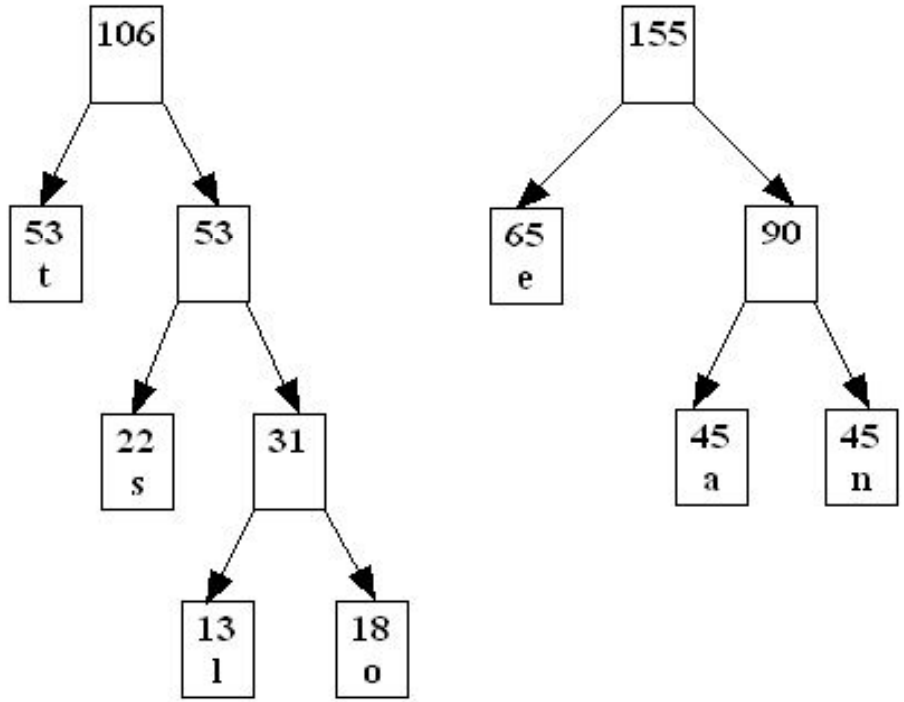


مثال کد هافمن

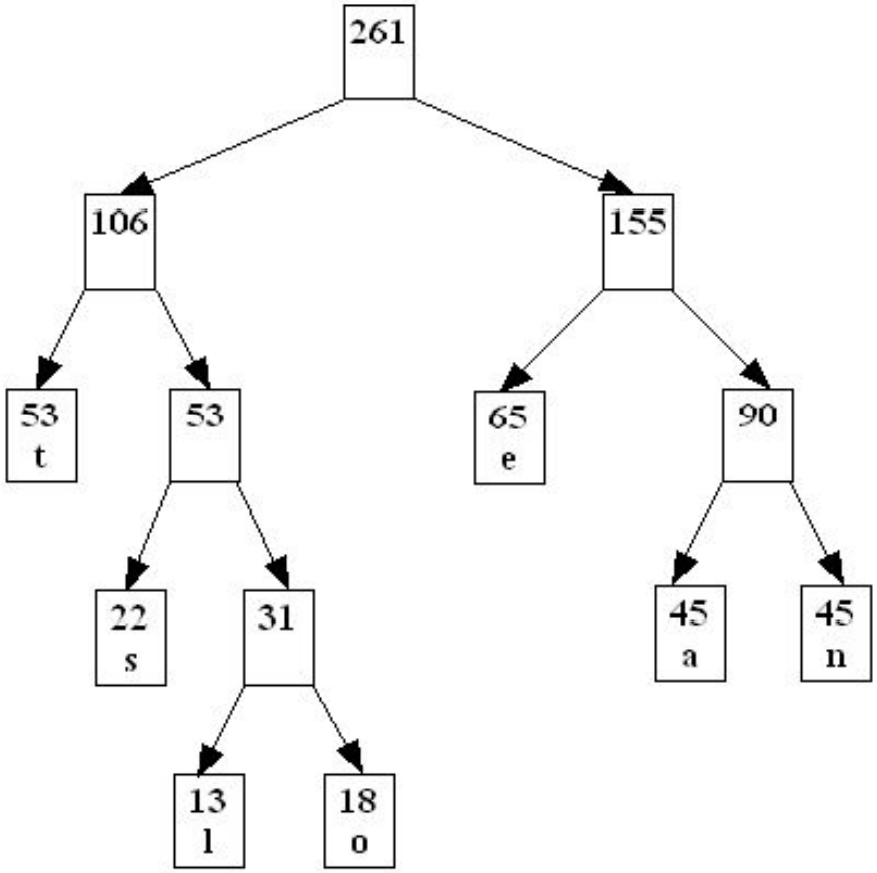




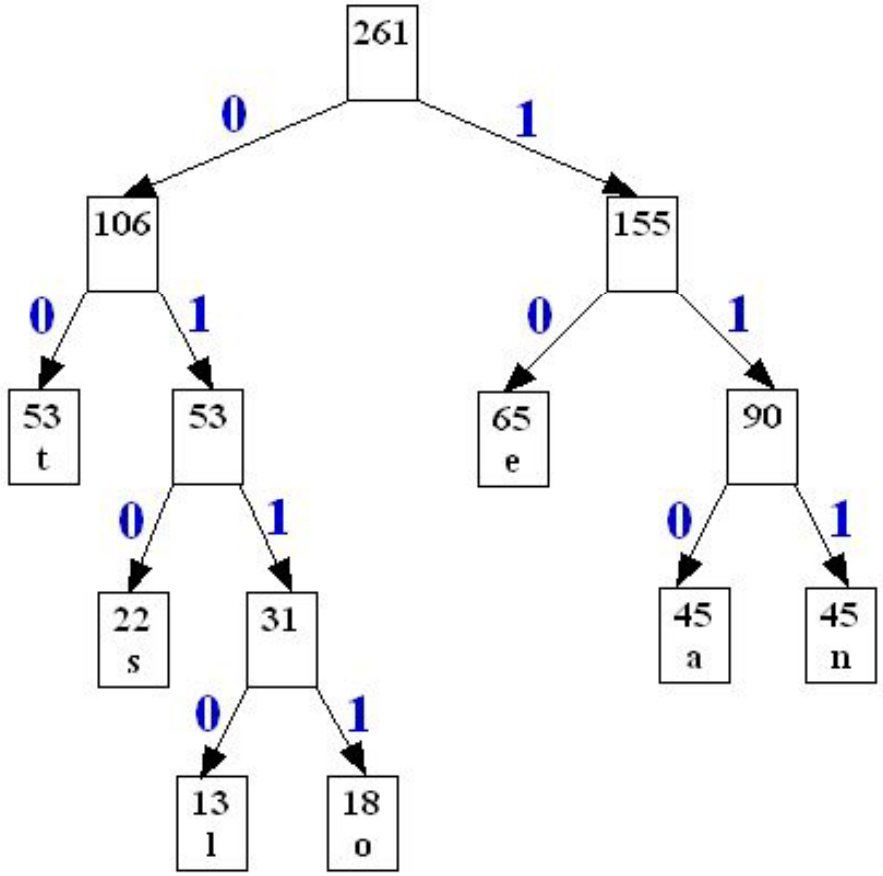
مثال کد هافمن



مثال کد هافمن



مثال کد هافمن



character	a	e	l	n	o	s	t
Huffman codeword	110	10	0110	111	0111	010	00

مثال کد هافمن

character	a	e	l	n	o	s	t	
codeword	110	10	0110	111	0111	010	00	
codeword bits	3	2	4	3	4	3	2	
character frequency	45	65	13	45	18	22	53	
codeword bits * frequency	135	130	52	135	72	66	106	sum 696

خصوصیات کدینگ هافمن

- **1- خصوصیت منحصر به‌فرد پیشوندی:** هیچ کد هافمنی پیشوند کد هافمن دیگر نیست. این نکته مانع ایجاد هر گونه ابهامی در مرحلهٔ رمزگشایی می‌شود.

- **2- بهینگی:** حداقل افزونگی کد.
  - دو نماد با کمترین فراوانی ، دارای طول مساوی برای کد هافمنشان هستند و تنها در آخرین بیت تفاوت دارند.
  - نمادهایی با فراوانی بیشتر دارای کد هافمن کوتاه‌تری نسبت به نمادهای با فراوانی کمتر هستند.
  - میانگین طول کد ، برای یک منبع اطلاعات S حتماً از $\eta+1$ کمتر است.

$$\bar{l} < \eta + 1$$

سوال

- مجموعه سمبل‌های موجود در سیستم شامل پنج عنصر متفاوت A ، B ، C ، D و E است. تعداد تکرار این سمبل‌ها در توالی ورودی به ترتیب 24 ، 12 ، 10 ، 8 و 8 مرتبه است.

- الف) با استفاده از روش Shannon-Fano کد هریک از سمبل‌ها را مشخص نمایید. و طول میانگین کد برای این روش را حساب کنید.

- ب) با استفاده از روش Huffman کد هریک از سمبل‌ها را مشخص نمایید. و طول میانگین کد برای این روش را حساب کنید.

- ج) انتروپی را برای این سیستم و مجموعه سمبل‌ها محاسبه کنید.

هدف

در این بخش نسخه‌های توسعه‌یافته و تطبیقی را ارائه خواهیم نمود.

کد هافمن توسعه‌یافته

- تمام کدهایی که در کدینگ هافمن تولید می‌شوند ، به طول صحیح فضا اشغال می‌کنند.اما همانطور که قبلا اشاره شد تعداد بیت های مورد نیاز برای هر کاراکتر از رابطه $\log_2 \frac{1}{p_i}$ به دست می آید این نشان می دهد زمانی که p_i خیلی بزرگ باشد (نزدیک به 1.0) $\log_2 \frac{1}{p_i}$ به سمت صفرمیل می‌کند و اختصاص یک بیت برای نشان دادن این کاراکتر منجر به هزینه در فضای ذخیره سازی خواهد شد.
- زمانی که احتمال حضور همه کاراکترها برابر 2^{-k} (K یک عدد صحیح مثبت) باشد می توان گفت که متوسط اندازه کد بهینه است.
- برای حل این مشکل کدینگ هافمن توسعه یافته مطرح گردیده است.

الفبای توسعه یافته:

در مجموعه زیر:

$$S = \{s1, s2, \ldots, sn\},$$

اگر k سمبل با یکدیگر تشکیل یک گروه را داده باشند الفبای توسعه یافته به صورت زیر است:

$$S^{(k)} = \{\overbrace{s_1 s_1 \ldots s_1}^{k\ symbols}, s_1 s_1 \ldots s_2, \ldots, s_1 s_1 \ldots s_n, s_1 s_1 \ldots s_2 s_1, \ldots, s_n s_n \ldots s_n\}.$$

کد هافمن توسعه‌یافته

رابطهٔ زیر برای به دست آوردن میانگین تعداد بیت‌های هر سمبل قابل اثبات است :

$$\eta \le \bar{l} < \eta + \frac{1}{k}$$

- که در مقایسه با کد نویسی هافمن اصلی بهبود یافته است ، البته نه به میزان زیاد.

- **مشکل:** اگر k نسبتاً بزرگ باشد (مثلاً برای $k \geq 3$) و $n \gg 1$ ، تعداد سمبل‌ها (n^k) عدد بسیار بزرگی می‌شود.

کد هافمن تطبیقی

- اطلاعات آماری سمبل‌ها به مرور جمع آوری شده و به صورت پویا به شکل یک رشته داده به‌روز رسانی می‌شوند.

```
ENCODER                    DECODER
-------                   -------
Initial_code();           Initial_code();

while not EOF             while not EOF
 {                        {
      get(c);                   decode(c);
      encode(c);                output(c);
      update_tree(c);           update_tree(c);
 }                        }
```

کد هافمن تطبیقی

- *Initial_code* بدون اطلاع از تعداد سمبل‌ها به آنها کدهای اولیه اختصاص می دهد.(مثلا کد اسکی)
- دستور *update_tree* یک درخت هافمن تطبیق یافته می‌سازد.

در واقع این دستور دو عمل انجام می‌دهد:

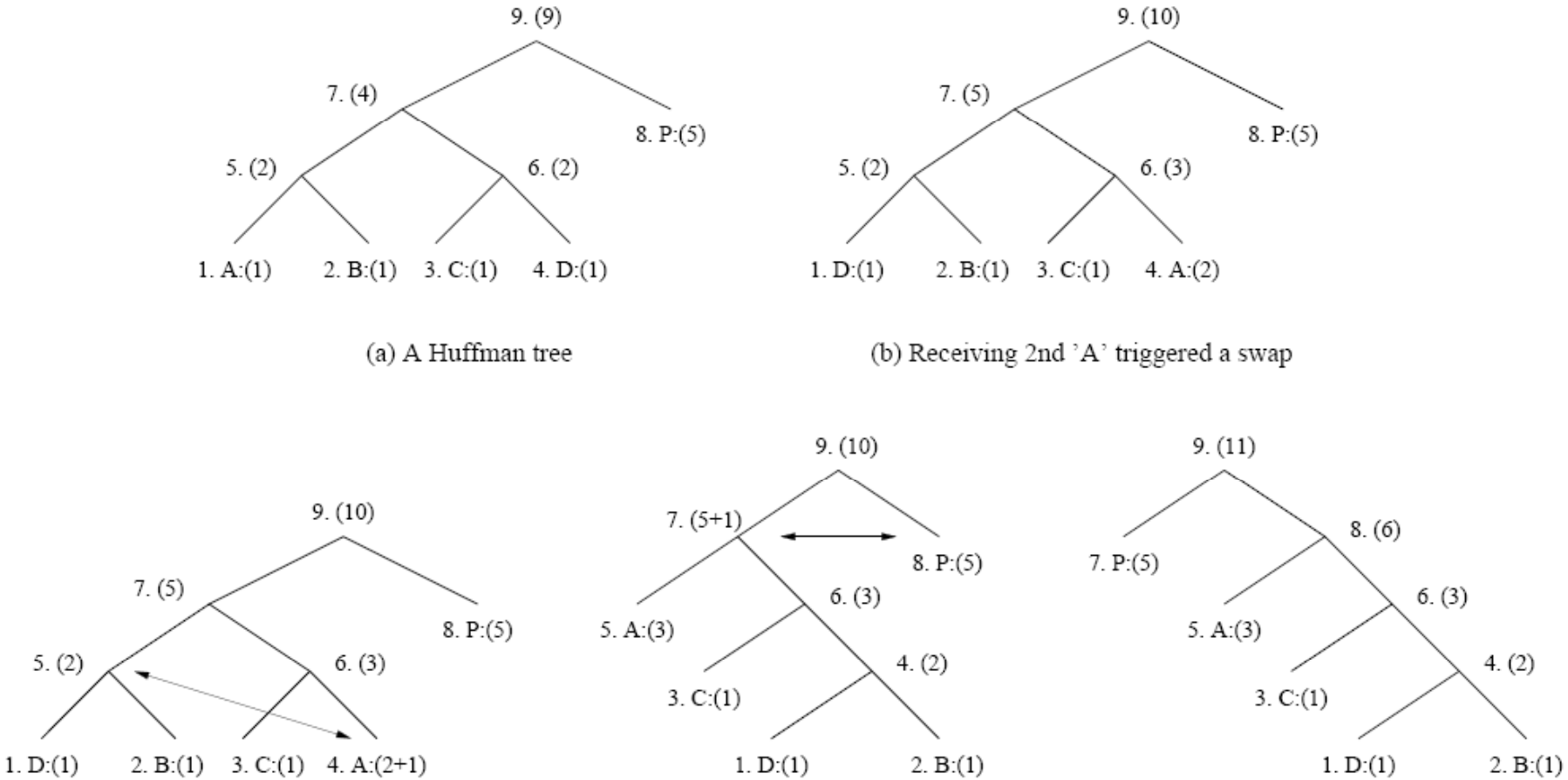
الف) تعداد (count) را برای سمبل‌ها یک واحد افزایش می‌دهد ، یا سمبلی جدید ایجاد می‌کند.

ب) وضعیت یا شکل درخت جدید را به‌روز رسانی می‌کند.

کد هافمن تطبیقی

- گره‌ها به ترتیب از چپ به راست و پایین به بالا شماره‌گذاری شده‌اند.
- درخت همیشه باید خاصیت *sibling* خود را حفظ کند. یعنی تمام گره‌ها (داخلی و برگ) باید بر اساس شماره خود وتعداد فراوانی به صورت صعودی مرتب باشند.(گره با شماره بیشتر دارای فراوانی بیشتر باشد و برعکس)
- اگر خاصیت *sibling* نقض شود رویه *swap* فراخوانی می‌شود تا درخت را مجدداً مرتب و به‌روزرسانی کند.
- هنگامی که استفاده از *swap* لازم می‌شود دورترین گره با مقدار Count=N با گره‌ای که Count آن برابر N+1 است جابجا می‌شود.
- *encoder* و *decoder* از روال‌های *Initial_code* و *update_tree* به طور کاملاً مشابهی استفاده می‌کنند.

کد هافمن تطبیقی



(a) A Huffman tree

(b) Receiving 2nd 'A' triggered a swap

(c−1) A swap is needed after receiving 3rd 'A'

(c−2) Another swap is needed

(c−3) The Huffman tree after receiving 3rd 'A'

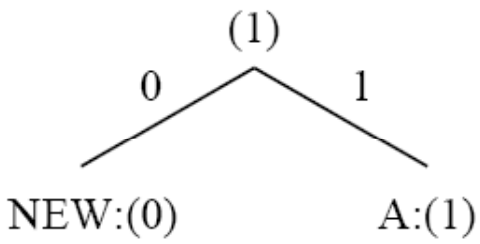
جابجایی گره ها برای به روز رسانی درخت هافمن تطبیق یافته

کد هافمن تطبیقی

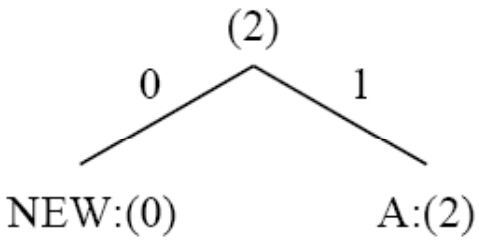
- هر کاراکتر یا سمبلی که برای اولین بار فرستاده می‌شود باید به وسیله نماد خاصی به نام NEW مشخص شود. مقدار اولیهٔ کد NEW صفر است. تعداد آن همیشه صفر نگه داشته می‌شود (و هرگز افزایش نمی‌یابد).
- مثال: کدگذاری رشتهٔ **AADCCDD**

```
      Initial Code
   --------------------
NEW:      0
A:        00001
B:        00010
C:        00011
D:        00100
          . .
          . .
          . .
```

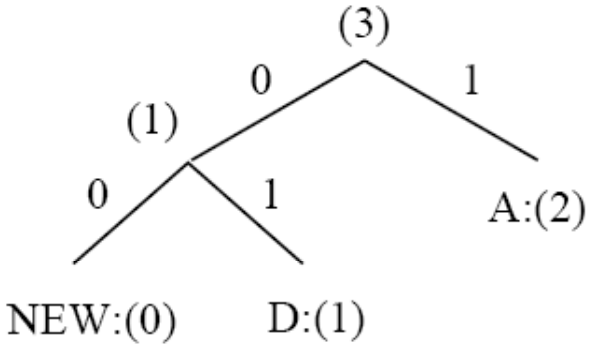
درخت هافمن تطبیقی برای AADCCDD



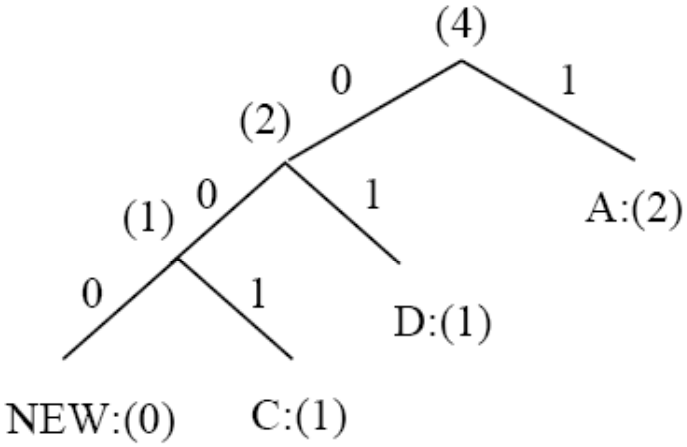
"A"



"AA"

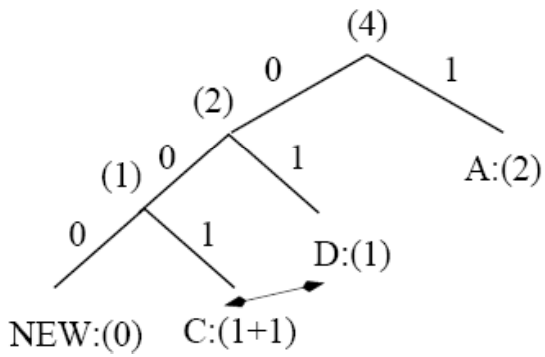


"AAD"

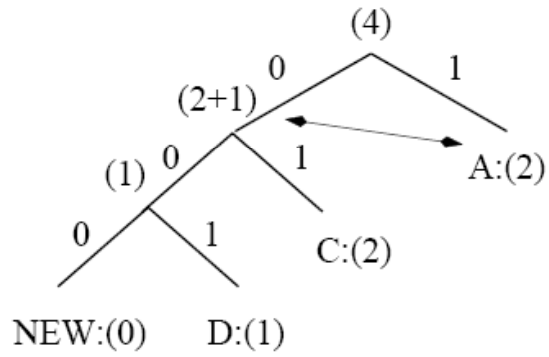


"AADC"

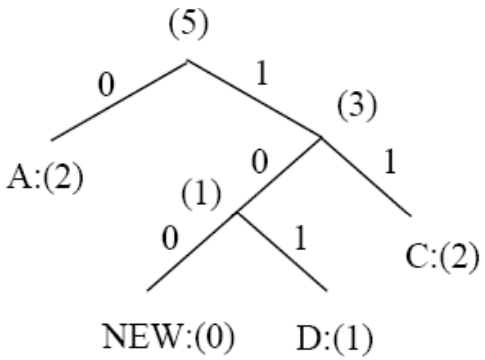
درخت هافمن تطبیق یافته برای AADCCDD



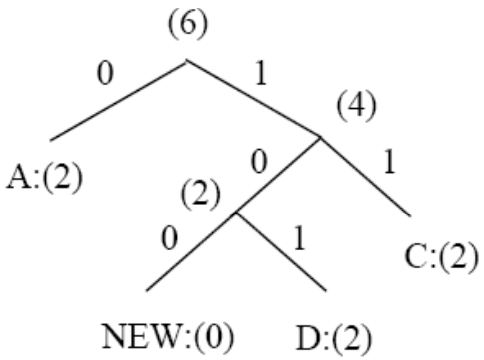
"AADCC" Step 1



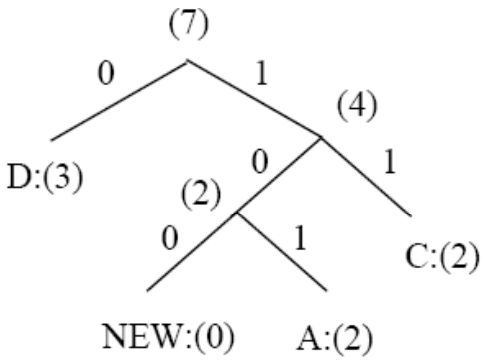
"AADCC" Step 2



"AADCC" Step 3



"AADCCD"



"AADCCDD"

کد هافمن تطبیقی

دنباله‌ای از سمبل‌ها و کدهایی که به decoder فرستاده می‌شوند:

Symbol	NEW	A	A	NEW	D	NEW	C	C	D	D
Code	0	00001	1	0	00100	00	00011	001	101	101

نکته: کدی که برای یک سمبل خاص در نظر گرفته شده ، در طول عملیات کدگذاری هافمن تطبیق یافته تغیر می‌کند.

هدف

در این بخش رویکرد کدگذاری مبتنی بر دیکشنری را ارائه خواهیم نمود.

کدگذاری بر پایه Dictionary

- LZW (Lempel–Ziv–Welch) از کلمه کدهایی با طول ثابت برای بیان کردن رشته‌ای از سمبل‌ها و کاراکترها با طول متغیر استفاده می‌کند.

- LZW در encoder و decoder هنگام دریافت داده ، dictionary یکسانی را به صورت پویا تولید می‌کند.

LZW Compression

```
BEGIN
   s = next input character;
   while not EOF
   {
          c = next input character;

          if s + c exists in the dictionary
          s = s + c;
          else
          {
          output the code for s;
          add string s + c to the dictionary with a new code;
          s = c;
          }
   }
   output the code for s;
END
```

کدگذاری بر پایه Dictionary

- مثال: کدکردن رشتهٔ "ABABBABCABABBA"

Code	String
1	A
2	B
3	C

کدگذاری بر پایه Dictionary

S	C	Output	Code	String
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA
A	B			
AB	A	4	10	ABA
A	B			
AB	B			
ABB	A	6	11	ABBA
A	EOF	1		

“ABABBABCABABBA”

کدهای خروجی عبارتند از: 1 6 4 3 2 5 4 2 1. به جای فرستادن 14 کاراکتر تنها
لازم است 9 کد فرستاده شود.(نرخ فشرده سازی : 1.56 = 14/9)

LZW Decompression

```
BEGIN
  s = NIL;
  while not EOF
  {
     k = next input code;
     entry = dictionary entry for k;
     output entry;
     if (s != NIL)
             add string s + entry[0] to dictionary with a new
  code;
     s = entry;
  }
END
```

کدگذاری بر پایه Dictionary

- **مثال : رمزگشایی** برای رشتهٔ "ABABBABCABABBA" .

کدهای ورودی به decoder: 1 2 4 5 2 3 4 6 1

S	K	Entry/output	Code	String
			1	A
			2	B
			3	C
NIL	1	A		
A	2	B	4	AB
B	4	AB	5	BA
AB	5	BA	6	ABB
BA	2	B	7	BAB
B	3	C	8	BC
C	4	AB	9	CA
AB	6	ABB	10	ABA
ABB	1	A	11	ABBA
A	EOF			

کدگذاری بر پایه Dictionary

- در برنامه‌های کاربردی واقعی ، l طول کد در بازهٔ $[l_0, l_{max}]$ قرار می‌گیرد.سایز دیکشنری در ابتدا 2^{l_0} است. وقتی که این دیکشنری تکمیل می‌شود ، طول کد یکی یکی افزایش می‌یابد ، این افزایش تا زمانی مجاز است که طول کد به l_{max} برسد.

- وقتی به l_{max} رسیدیم ، دیکشنری پر شده و می‌بایست آن را خالی نمائیم.
  - الگوریتم LRU

تمرین

- رشتهٔ ABRACADABRA را با استفاده از الگوریتم LZW کد کنید. تمامی فرض‌های مورد نیاز را اتخاذ کرده و آن‌ها را بیان کنید.

هدف

در این جزء الگوریتم کدگذاری حسابی را مطرح خواهیم کرد.

کدگذاری حسابی

- تمام کدهایی که در کدینگ هافمن تولید می‌شوند ، به طول صحیح فضا اشغال می‌کنند.اما تعداد بیت های مورد نیاز برای هر کاراکتر از رابطه $\log_2 \frac{1}{p_i}$ به دست می آید این نشان می دهد زمانی که p_i خیلی بزرگ باشد (نزدیک به 1.0) $\log_2 \frac{1}{p_i}$ به سمت صفرمیل می‌کند و اختصاص یک بیت برای نشان دادن این کاراکتر منجر به هزینه در فضای ذخیره سازی خواهد شد.
- کدگذاری حسابی تمام پیام را به صورت یک داده در نظر می گیرد و آن را کد می کند.
- هر پیام با یک عدد در بازه [a,b) مشخص می شود. در شروع کار a و b برابر 0و1 هستند یعنی بازه [0,1)
- وقتی پیام بزرگتر می شود اندازه بازه کوچکتر می شود و تعداد بیت های لازم برای نمایش بازه افزایش می یابد.

کدگذاری حسابی

مثال: رمز نمادهای "CAEE$" با استفاده از کدگذاری حسابی

در این مثال حروف الفبا [A,B,C,D,E,F,$]هستند. که $ یک نماد ویژه برای نمایش پایان پیام است.

جدول احتمال حضور هر حرف به صورت زیر است.

نماد	احتمال	محدوده
A	0.2	[0, 0.2)
B	0.1	[0.2, 0.3)
C	0.2	[0.3, 0.5)
D	0.05	[0.5, 0.55)
E	0.3	[0.55, 0.85)
F	0.05	[0.85, 0.9)
$	0.1	[0.9, 1.0)

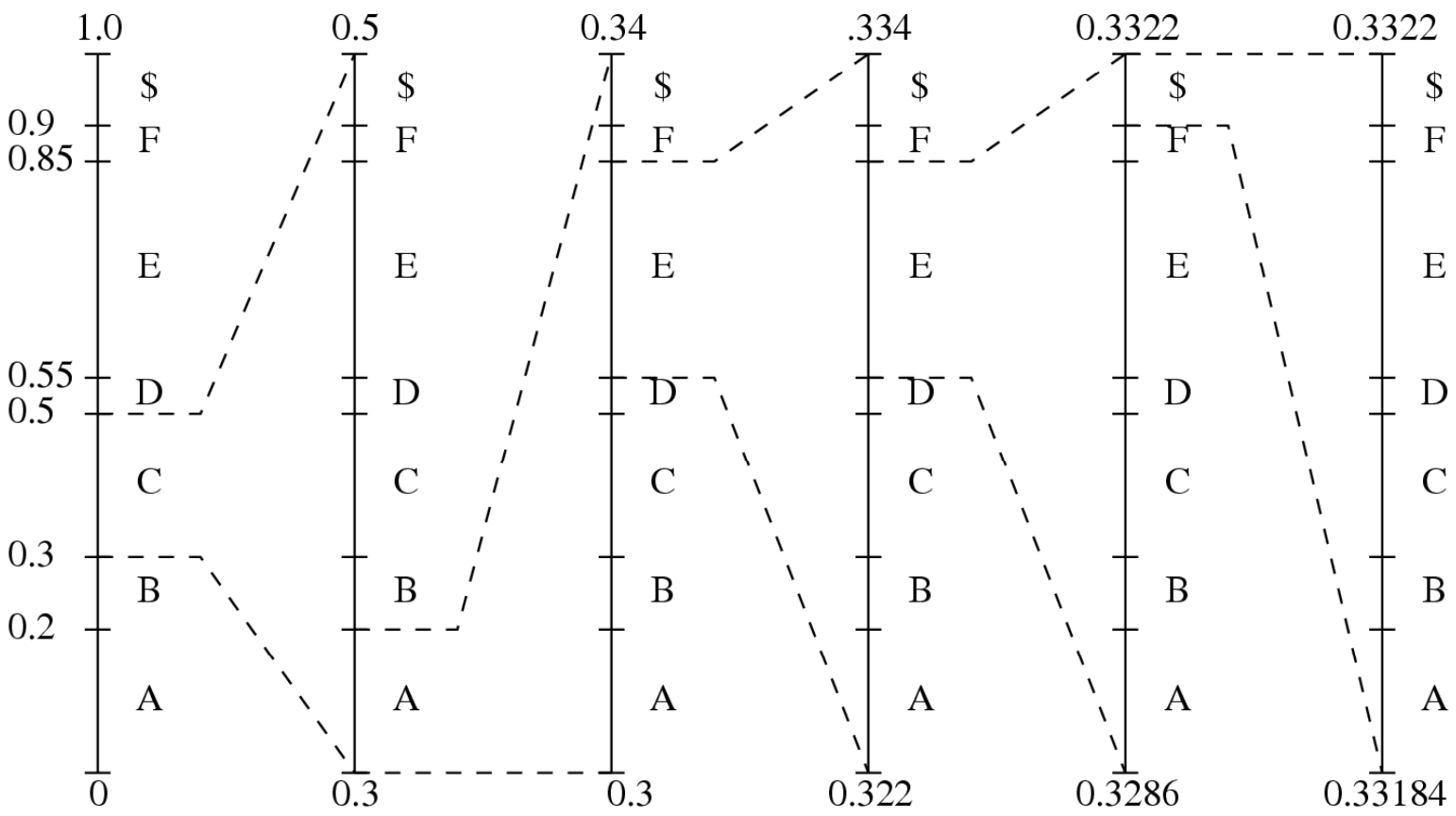
Arithmetic Coding Encoder

```
BEGIN
  low = 0.0;        high = 1.0;  range = 1.0;

  while (symbol != terminator)
  {
    get (symbol);
    low = low + range * Range_low(symbol);
    high = low + range * Range_high(symbol);
    range = high - low;
  }

  output a code so that low <= code < high;
END
```

کدگذاری حسابی



کدگذاری حسابی

کدگذاری حسابی رشتهٔ "CAEE$"

Symbol	Low	High	Range
	0	1.0	1.0
C	0.3	0.5	0.2
A	0.30	0.34	0.04
E	0.322	0.334	0.012
E	0.3286	0.3322	0.0036
$	0.33184	0.33220	0.00036

کدگذاری حسابی

کدگذاری حسابی رشتهٔ "CAEE$".

برای شروع low=0, high=1, range=1

پس از رسیدن اولین کاراکتر C چون range_low(c)=0.3, range_high(c)=0.5 پس low=0+1.0*0.3=0.3 و high=0+1.0*0.5=0.5 می شود و در نتیجه مقدار range به 0.2 کاهش پیدا می کند. این روند همانند شکل - ادامه پیدا می کند تا زمانی که کاراکتر$ برسد. در این جا low= 0.33184 و high= 0.33220 و range= 0.00036 است.

واضح است که برای محاسبه range می توان از رابطه زیر هم استفاده کرد:

$$range = P_C \times P_A \times P_E \times P_E \times P_S = 0.2 \times 0.2 \times 0.3 \times 0.3 \times 0.1 = 0.00036$$

کدگذاری حسابی

تولید کلمهٔ کد برای کدکننده (Codeword)

```
BEGIN
  code = 0;
  k = 1;
  while (value(code) < low)
  {
    assign 1 to the kth binary fraction bit
    if (value(code) > high)
    replace the kth bit by 0
    k = k + 1;
  }
END
```

- تولید عددی در بازه ، (حد بالا ،حد پایین)
- به عنوان نمونه اگر بازهٔ انتخاب شده [0.33184, 0.33220] باشد ، کد خروجی می‌شود 0.01010101 خواهد بود.

کدگذاری حسابی

- به عنوان نمونه اگر بازهٔ انتخاب شده [0.33184, 0.33220] باشد.
- اگر مقدار 1 را به اولین بیت اختصاص دهیم ، عدد برابر 0.1 در مبنای دو خواهد شد که این عدد در مبنای ده برابر 0.5 است وچون این مقدار از high بیشتر است پس این بیت برابر صفر خواهد شد. و کد برابر 0.0 است
- چون مقدار 0.0 در مبنای ده کوچکتر از low است پس حلقه ادامه پیدا می کند.
- در مرحله بعد با اختصاص 1 به بیت بعدی داریم 0.01 که این عدد در مبنای 10 برابر 0.25 است و چون این مقدار کوچکتر از high است پس تغیر نمی کند.
- چون مقدار 0.01 در مبنای ده کوچکتر از low است پس حلقه ادامه پیدا می کند.

کدگذاری حسابی

- در مرحله بعد با اختصاص 1 به بیت بعدی داریم 0.011 که این عدد در مبنای 10 برابر 0.375 است وچون این مقدار از high بیشتر است پس این بیت برابر صفر خواهد شد. و کد برابر 0.010است
- چون مقدار 0.010 در مبنای ده کوچکتر از low است پس حلقه ادامه پیدا می کند.

- در ادامه با اختصاص 1 به بیت بعدی داریم 0.0101 که این عدد در مبنای 10 برابر 0.3125 است و چون این مقدار کوچکتر از high است پس تغیر نمی کند.
- چون مقدار 0.0101 در مبنای ده کوچکتر از low است پس حلقه ادامه پیدا می کند.

- به همین ترتیب کد تولید شده برای بازه $[0.33184, 0.33220]$ برابر 0.01010101 خواهد بود.
- این کد برابر $2^{-2}+2^{-4}+2^{-6}+2^{-8}$ است. و 8 بیت برای ذخیره آن نیاز است.
- در حالت عادی چون $\log_2 \frac{1}{0.00036} \approx 11.44$ ممکن بود تا دوازده بیت هم الگوریتم ادامه پیدا کند.

کدگذاری حسابی

Arithmetic Coding Decoder:

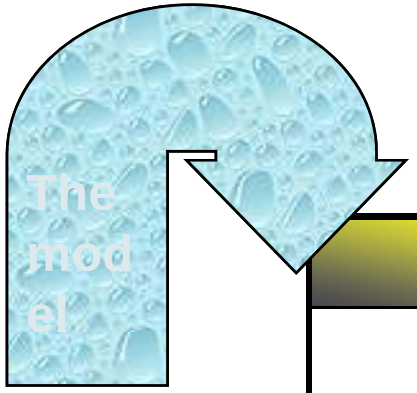
```
BEGIN
  get binary code and convert to
  decimal value = value(code);
  Do
  {
    find a symbol s so that
       Range_low(s) <= value < Range_high(s);
    output s;
    low = Rang_low(s);
    high = Range_high(s);
    range = high - low;
    value = [value - low] / range;
  }
  Until symbol s is a terminator
END
```

کدگذاری حسابی

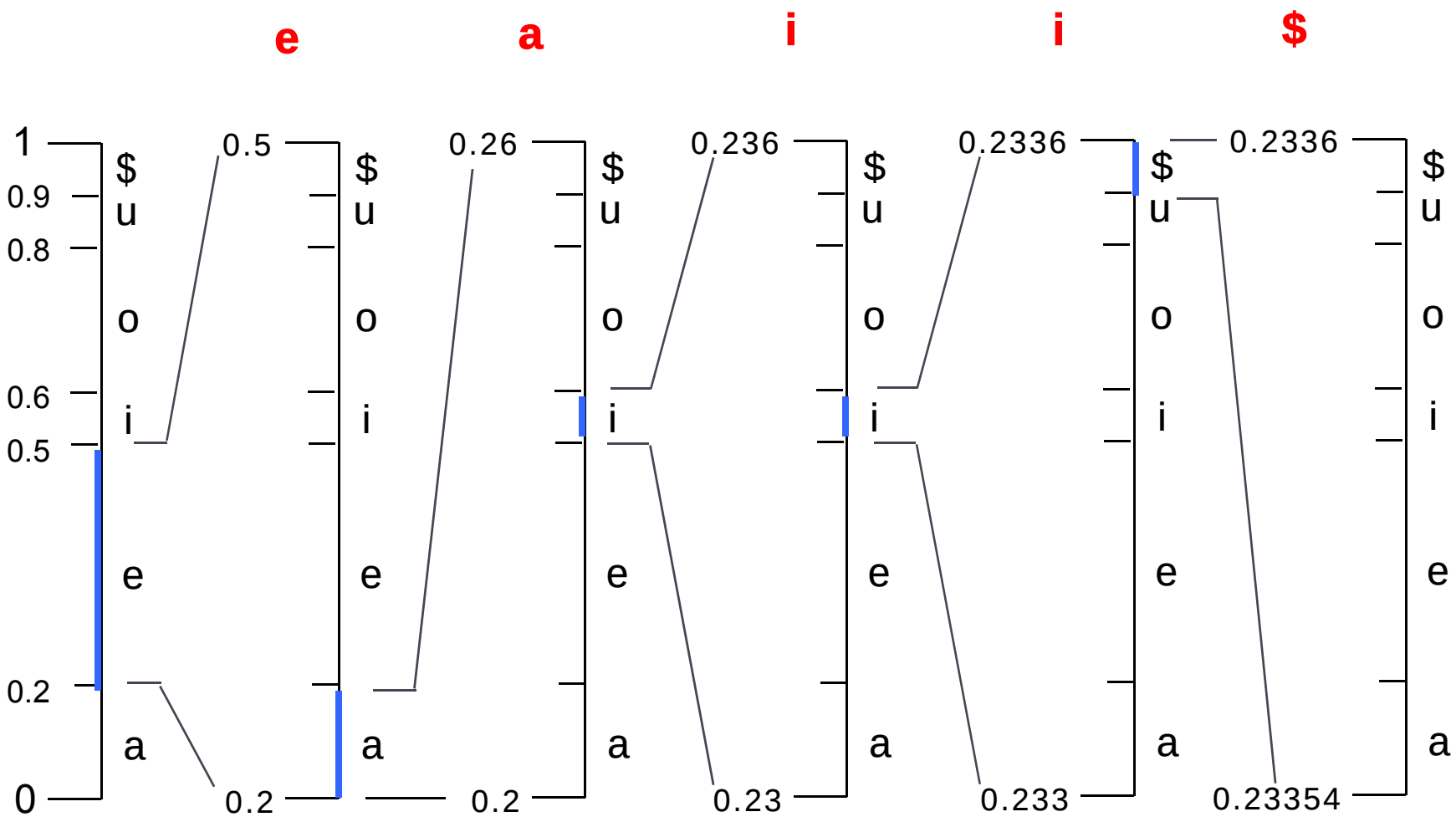
رمز گشایی نمادهای "CAEE$"

Value	نماد خروجی	حد پایین	حدبالا	محدوده
0.33203125	C	0.3	0.5	0.2
0.16015625	A	0.0	0.2	0.2
0.80078125	E	0.55	0.85	0.3
0.8359375	E	0.55	0.85	0.3
0.953125	$	0.9	1.0	0.1

مثالی دیگر رشته eaii$



Symbol	Probability	Range
a	.2	[0,0.2)
e	.3	[0.2, 0.5)
i	.1	[0.5, 0.6)
o	.2	[0.6, 0.8)
u	.1	[0.8, 0.9)
$	.1	[0.9, 1)


e
a
i
i
$
1
0.9
0.8
0.6
0.5
0.2
0
$
u
o
i
e
a
0.5
0.2
0.26
0.2
0.236
0.23
0.2336
0.233
0.2336
0.23354

- اندازه بازه نهایی برابر است با
0.2336 -0.23354 = **0.00006**,

(0.3)* (0.2)* (0.1)* (0.1)* (0.1) = **0.00006**.