

VERİ SIKIŞTIRMA

İstatistiksel Yöntemler-5

Prof.Dr. Banu DİRİ

Move -To -Front Kodlama

- Bölgesel adaptif bir entropy kodlama yöntemidir
- Eldeki mevcut alfabe kullanılarak, input stream'de yer alan karakterlere kullanım sıklıklarına bağlı olarak bir kod verilir
- Yöntem gereği sıklıklar önceden hesaplanmaz
- Sıkıştırma performansını artırmak için, öncesinde kullanılan bir kodlamadır

Input stream a b c d d c b a m n o p p o n m

alfabe {abcdmnop}

Code 0 1 2 3 0 1 2 3 4 5 6 7 0 1 2 3

Input move-to-front code

a	abcdmnop	0
b	abcdmnop	1
c	bacdmnop	2
d	cbadmno	3
d	dcbamno	0
c	dcbamno	1
b	cdbamno	2
a	bcdamno	3
m	abcdmnop	4
n	mabcdnop	5
o	nmabcdop	6
p	onmabcdp	7
p	ponmabcd	0
o	ponmabcd	1
n	opnmabcd	2
m	nopmabcd	3
	mnopabcd	

Entropy = H = 2.81 bits/char
Output stream deki her code ortalama 2.81 bit ile kodlanır.



Kod Çözme İşlemi

Kod çözme işleminde Input tersten okunur ve alfabenin ilk hali oluşturulur

Code içerisindeki değerleri kodlamak için Huffman veya Aritmetik Kodlama kullanılır

	m n o p a b c d
m	m n o p a b c d
n	n m o p a b c d
o	o n m p a b c d
p	p o n m a b c d
p	p o n m a b c d
o	o p n m a b c d
n	n o p m a b c d
m	m n o p a b c d
a	a m n o p b c d
b	b a m n o p c d
c	c b a m n o p d
d	d c b a m n o p
d	d c b a m n o p
c	c d b a m n o p
b	b c d a m n o p
a	a b c d a m n o p

Aynı işlemi Move-To-Front kodlama kullanmadan yapalım

Input stream **a b c d d c b a m n o p p o n m**

alfabe {abcdmnop}

Input move-to-front code

a	abcdmnop	0
b	abcdmnop	1
c	abcdmnop	2
d	abcdmnop	3
d	abcdmnop	3
c	abcdmnop	2
b	abcdmnop	1
a	abcdmnop	0
m	abcdmnop	4
n	abcdmnop	5
o	abcdmnop	6
p	abcdmnop	7
p	abcdmnop	7
o	abcdmnop	6
n	abcdmnop	5
m	abcdmnop	4

Entropy = $H = 3$ bits/char
Output stream deki her code ortalama 3 bit ile kodlanır.



Kod Çözme İşlemi

Kod çözme işleminde Input tersten okunur ve alfabenin ilk hali oluşturulur

Code içerisindeki değerleri kodlamak için Huffman veya Aritmetik Kodlama kullanılır

Entropy nin daha yüksek çıkmasının sebebi arka arkaya tekrar eden kod olmamasıdır.

	a b c d m n o p
m	a b c d m n o p
n	a b c d m n o p
o	a b c d m n o p
p	a b c d m n o p
p	a b c d m n o p
o	a b c d m n o p
n	a b c d m n o p
m	a b c d m n o p
a	a b c d m n o p
b	a b c d m n o p
c	a b c d m n o p
d	a b c d m n o p
d	a b c d m n o p
c	a b c d m n o p
b	a b c d m n o p
a	a b c d m n o p

Move-To-Front yönteminde sadece karakterleri değil kelimeleri de kodlayabiliriz.

the boy on my right is the right boy

Word	A (before adding)	A (after adding)	Code emitted
the	()	(the)	0the
boy	(the)	(the, boy)	1boy
on	(boy, the)	(boy, the, on)	2on
my	(on, boy, the)	(on, boy, the, my)	3my
right	(my, on, boy, the)	(my, on, boy, the, right)	4right
is	(right, my, on, boy, the)	(right, my, on, boy, the, is)	5is
the	(is, right, my, on, boy, the)	(is, right, my, on, boy, the)	5
right	(the, is, right, my, on, boy)	(the, is, right, my, on, boy)	2
boy	(right, the, is, my, on, boy)	(right, the, is, my, on, boy)	5
	(boy, right, the, is, my, on)		

Move-To-Front yönteminde decode işlemi

<u>Code</u>	<u>input</u>	<u>before adding</u>	<u>after adding</u>	<u>decode word</u>
0	the	()	(the)	the
1	boy	(the)	(the, boy)	boy
2	on	(boy, the)	(boy, the, on)	on
3	my	(on, boy, the)	(on, boy, the, my)	my
4	right	(my, on, boy, the)	(my, on, boy, the, right)	right
5	is	(right, my, on, boy, the)	(right, my, on, boy, the, is)	is
5		(is, right, my, on, boy, the)	(is, right, my, on, boy, the)	the
2		(the, is, right, my, on, boy)	(the, is, right, my, on, boy)	right
5		(right, the, is, my, on, boy)	(right, the, is, my, on, boy)	boy
		(boy, right, is, my, on, the)		

İstatistiksel Yöntemler

- ☐ Prefix Kodlar
- ☐ Shannon Fano Coding
- ☐ Huffman Coding
- ☐ Arithmetic Coding
- ☐ Diğer Yöntemler

Değişken Uzunluktaki Kodlar (Variable Size Code)

{a1, a2, a3, a4} hepsi eşit olasılığa sahip olsun ($P_i=0,25$)

$$H = -(4 * 0,25 \log_2 0,25) = 2 \text{ bits}$$

Her sembolü kodlamak için ortalama 2 bit yeterli olmaktadır.

(00 01 10 11)

{a1, a2, a3, a4} sembollerin farklı olasılık değerlerine sahip olduğunu düşünelim.

$$P(a1) = 0,49 \quad P(a2) = 0,25 \quad P(a3) = 0,25 \quad P(a4) = 0,01$$

$$H = - (0,49 \log_2 0,49 + 0,25 \log_2 0,25 + 0,25 \log_2 0,25 + 0,01 \log_2 0,01) = 1,57 \text{ bits}$$

$$\text{Redundancy} = R = |2 - 1,57| = 0,43$$

Input {a1 a3 a2 a1 a3 a3 a4 a2 a1 a1 a2 a2 a1 a1 a3 a1 a1 a2 a3 a1}

Sembol	Olasılık	Code1	code2
a1	0,49	1	1
a2	0,25	01	01
a3	0,25	010	000
a4	0,02	001	001

Code1 ile kodlandığında

9 a1 → 1 bit

5 a2 → 2 bit

5 a3 → 3 bit

1 a4 → 3 bit

20 sembol 37 bit ile kodlanıyor.

Ortalama Mesaj Uzunluğu = $S = 37/20 = 1,85$ bit

$H = 1,57$

$R = |H - S| = 0,28$

Sıkıştırma performansı iyi, ancak bu kodun geri dönüşü yoktur.

NEDEN ?

Sembol	Olasılık	Code1	code2
a1	0,49	1	1
a2	0,25	01	01
a3	0,25	010	000
a4	0,02	001	001

Code2 kullanmak gerekir. Code2 **prefix** (ön ek) özelliğine sahiptir.

Prefix Code : Bir sembolün sahip olduğu kodun bit pattern, diğer sembolün ön eki olamaz.

Değişken uzunlukta kod oluşturulurken iki temel özelliğe dikkat edilmelidir.

- En fazla sıklıkta kullanılan sembole, en kısa kod atanır
- Prefix özelliğine uyulmalıdır

Integer sayıların binary gösterilimi prefix kod özelliğine uyar mı?

HAYIR

Prefix Kod Örnekleri

$i \geq 1$ olmak şartı ile, i .kodun binary değerinin başlangıcında $\lfloor \log_2 i \rfloor$ adet sıfır eklenir.

i	code	size
1	1	1
2	010	3
3	011	3
4	00100	5
5	00101	5
6	00110	5
7	00111	5
8	0001000	7
....
15	0001111	7
16	000010000	9

i kodunun toplam bit uzunluğu nedir ?

$$1 + 2 \lfloor \log_2 i \rfloor \quad \text{bits}$$

UNARY CODE

Negatif olmayan n integer sayısının, **unary kodu** $(n-1)$ adet 1 ve onları takip eden bir tane 0'dan oluşmaktadır veya $(n-1)$ tane 0 ve onları takip eden bir tane 1'den oluşmaktadır.

Örnek : 680 adet unary kodu nasıl elde edebiliriz. 680 adet kod için triplet (3,2,9)

n	code	Alter code
1	0	1
2	10	01
3	110	001
4	1110	0001
5	11110	00001

n	a=3+n*2	n th codeword	#codeword	Integer sınır
0	3	0xxx	$2^3 = 8$	0-7
1	5	10xxxxx	$2^5 = 32$	8-39
2	7	110xxxxxxxx	$2^7 = 128$	40-167
3	9	111xxxxxxxxxx	$2^9 = 512$	168-679

- Unary kodlar, «start-step-stop» kod olarak da bilinir.
- $a = \text{start} + n * \text{step}$
- Eğer $a = \text{stop}$ ise a bitinin önündeki 0 düşürülür.

Start, step ve stop değerlerine bağlı olarak kaç adet unary kod oluşturulur?

$$\frac{2^{\text{stop}+\text{step}} - 2^{\text{start}}}{2^{\text{step}-1}} = \frac{2^{11} - 2^3}{2^2 - 1} = 680$$

Stop parametresine bağlı olarak üstel artar

Diğer Prefix Kodlar

n	$B(n)$	$\overline{B}(n)$	C_1	C_2	C_3	C_4
1	1		0	0	0	0
2	10	0	10 0	100	100 0	10 0
3	11	1	10 1	110	100 1	11 0
4	100	00	110 00	10100	110 00	10 100 0
5	101	01	110 01	10110	110 01	10 101 0
6	110	10	110 10	11100	110 10	10 110 0
7	111	11	110 11	11110	110 11	10 111 0
8	1000	000	1110 000	1010100	10100 000	11 1000 0
9	1001	001	1110 001	1010110	10100 001	11 1001 0
10	1010	010	1110 010	1011100	10100 010	11 1010 0
11	1011	011	1110 011	1011110	10100 011	11 1011 0
12	1100	100	1110 100	1110100	10100 100	11 1100 0
13	1101	101	1110 101	1110110	10100 101	11 1101 0
14	1110	110	1110 110	1111100	10100 110	11 1110 0
15	1111	111	1110 111	1111110	10100 111	11 1111 0
16	10000	0000	11110 0000	101010100	10110 0000	10 100 10000 0
31	11111	1111	11110 1111	111111110	10110 1111	10 100 11111 0
32	100000	00000	111110 00000	10101010100	11100 00000	10 101 100000 0
63	111111	11111	111110 11111	11111111110	11100 11111	10 101 111111 0
64	1000000	000000	1111110 000000	1010101010100	11110 000000	10 110 1000000 0
127	1111111	111111	1111110 111111	1111111111110	11110 111111	10 110 1111111 0
128	10000000	0000000	11111110 0000000	101010101010100	1010100 0000000	10 111 10000000 0
255	11111111	1111111	11111110 1111111	111111111111110	1010100 1111111	10 111 11111111 0

$B(n)$: n sayısının binary karşılığı
 $|B(n)|$: Binary sayısının dijit uzunluğu
 $\overline{B}(n)$: En soldaki bit atıldığında (her zaman 1) kalan bitler

C_1 kodu nasıl elde edilir ?

n	$B(n)$	$\overline{B}(n)$	C_1
1	1		0
2	10	0	10 0
3	11	1	10 1
4	100	00	110 00
5	101	01	110 01
6	110	10	110 10
7	111	11	110 11
8	1000	000	1110 000
9	1001	001	1110 001
10	1010	010	1110 010
11	1011	011	1110 011
12	1100	100	1110 100
13	1101	101	1110 101
14	1110	110	1110 110
15	1111	111	1110 111
16	10000	0000	11110 0000
31	11111	1111	11110 1111
32	100000	00000	111110 00000
63	111111	11111	111110 11111
64	1000000	000000	1111110 000000
127	1111111	111111	1111110 111111
128	10000000	0000000	11111110 0000000
255	11111111	1111111	11111110 1111111

- C_1 iki parçadan oluşur
- n değerinin binary karşılığı olan $B(n)$ nin dijit sayısı değerinin unary karşılığı ile $B(n)$ nin en soldaki bitinin atılmış hali olan $\overline{B}(n)$ nin birleşimidir

$n = 16$
 $(10000)_2$
 $B(n) = 10000$
 $\overline{B}(n) = 0000$
 $|B(n)| = 5$

$C_1 \rightarrow |B(n)| \mid \overline{B}(n)$

$C_1 \rightarrow 11110 \mid 0000$

veya

$C_1 \rightarrow 00001 \mid 0000$

$n=5$ unary code **11110**

$C_1(n)$ kod uzunluğu

$2 \lfloor \log_2 n \rfloor + 1$ bits

C_2 kodu nasıl elde edilir ?

n	$B(n)$	$\overline{B}(n)$	C_2
1	1		0
2	10	0	100
3	11	1	110
4	100	00	10100
5	101	01	10110
6	110	10	11100
7	111	11	11110
8	1000	000	1010100
9	1001	001	1010110
10	1010	010	1011100
11	1011	011	1011110
12	1100	100	1110100
13	1101	101	1110110
14	1110	110	1111100
15	1111	111	1111110
16	10000	0000	101010100
31	11111	1111	111111110
32	100000	00000	10101010100
63	111111	11111	11111111110
64	1000000	000000	1010101010100
127	1111111	111111	1111111111110
128	10000000	0000000	101010101010100
255	11111111	1111111	111111111111110

- C_1 kodunu yeniden düzenlenmesi ile oluşur
- C_1 'in birinci bölümündeki her bit teker teker alınırken, her bitin arkasına ikinci bölümden bir bit yazılır.

$n = 16$
 $(10000)_2$
 $B(n) = 10000$
 $\overline{B}(n) = 0000$
 $|B(n)| = 5$

$C_1 \rightarrow 11110 \mid 0000$

$C_2 \rightarrow 101010100$

$C_2(n)$ kod uzunluğu $2 \lfloor \log_2 n \rfloor + 1$ bits

C_3 kodu nasıl elde edilir ?

n	$B(n)$	$\bar{B}(n)$	C_3
1	1		0
2	10	0	100
3	11	1	100
4	100	00	110
5	101	01	110
6	110	10	110
7	111	11	110
8	1000	000	10100
9	1001	001	10100
10	1010	010	10100
11	1011	011	10100
12	1100	100	10100
13	1101	101	10100
14	1110	110	10100
15	1111	111	10100
16	10000	0000	10110
31	11111	1111	10110
32	100000	00000	11100
63	111111	11111	11100
64	1000000	000000	11110
127	1111111	111111	11110
128	10000000	0000000	1010100
255	11111111	1111111	1010100

- n 'nin binary gösteriliminin dijit sayısı alınır
- $|B(n)|$ nin değeri, C_2 kodu içerisinde bulunur ve daha sonra arkasına $\bar{B}(n)$ eklenir.

$C_3(15)$

1111 $\rightarrow |B(15)| = 4$

$C_2(4) = 10100 \mid \bar{B}(15) = 10100 \mid 111$

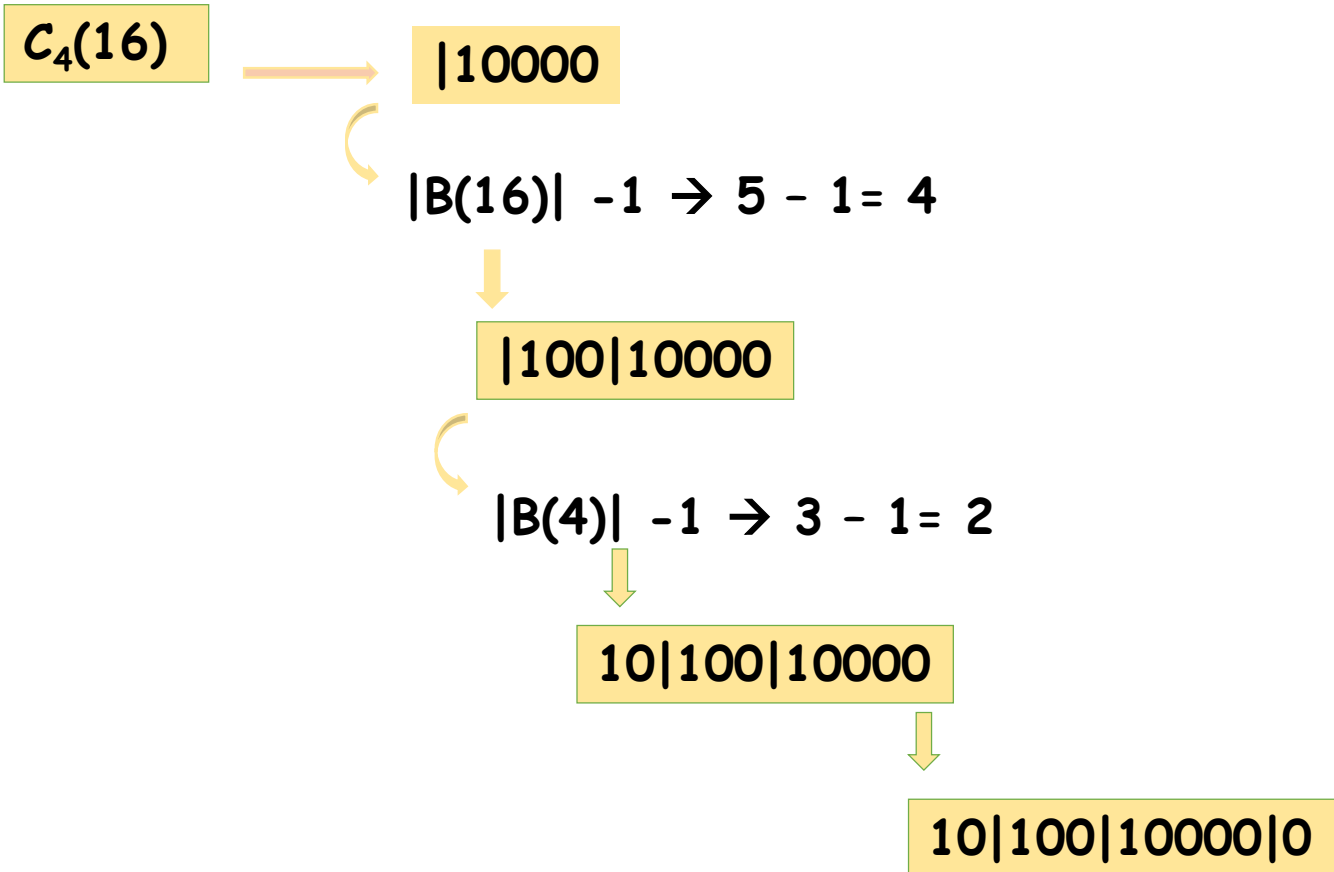
$C_3(n)$ kod uzunluğu

$1 + \lfloor \log_2 n \rfloor + 2 \lfloor \log_2 (1 + \lfloor \log_2 n \rfloor) \rfloor$ bits

C_4 kodu nasıl elde edilir ?

n	$B(n)$	$\overline{B}(n)$	C_4
1	1		0
2	10	0	10 0
3	11	1	11 0
4	100	00	10 100 0
5	101	01	10 101 0
6	110	10	10 110 0
7	111	11	10 111 0
8	1000	000	11 1000 0
9	1001	001	11 1001 0
10	1010	010	11 1010 0
11	1011	011	11 1011 0
12	1100	100	11 1100 0
13	1101	101	11 1101 0
14	1110	110	11 1110 0
15	1111	111	11 1111 0
16	10000	0000	10 100 10000 0
31	11111	1111	10 100 11111 0
32	100000	00000	10 101 100000 0
63	111111	11111	10 101 111111 0
64	1000000	000000	10 110 1000000 0
127	1111111	111111	10 110 1111111 0
128	10000000	0000000	10 111 10000000 0
255	11111111	1111111	10 111 11111111 0

- C_4 birden fazla bölümden oluşur
- İlk olarak hesaplanacak olan n kodunun binary gösterilimi yazılır
- Sol tarafına $|B(n)|-1$ değerinin binary gösterilimi yazılır
- İşleme $|B(n)|-1$ değeri iki dijit kalıncaya kadar devam ettirilir
- En son adımda da sağ tarafa 0 yazılır



GOLOMB CODE

Unary kodlama yönteminin özellikleri de kullanılarak oluşturulan basit bir prefix koddur.

- Kodları oluşturulacak alfabe içerisinde yer alan karakterler, kullanım frekansları/olasılıkları büyükten küçüğe doğru sıralanır.
- x olarak adlandırılan karakterlere 0 dan başlamak üzere, ardışık değerler atanır.
- İdeali 2'nin kuvveti olacak bir M değeri seçilir.
- Her x değeri için xM ve xL değerleri hesaplanır.
- Golomb kodlar, xM sayısının unary kod değeri ile xL sayısının binary değeri birleştirilerek elde edilir.

$$xM = \left\lfloor \frac{x}{m} \right\rfloor$$

$$xL = x \bmod M$$

$$x = (xM * M) + xL$$

$M=4$

x	xM	xL	Unary xM	Binary xL	Golomb Code
0	0	0	0	00	000
1	0	1	0	01	001
2	0	2	0	10	010
3	0	3	0	11	011
4	1	0	10	00	1000
5	1	1	10	01	1001
6	1	2	10	10	1010
7	1	3	10	11	1011
8	2	0	110	00	11000

Shannon Fano Coding

- ❖ Değişken uzunluklu bir kodlama yöntemidir
- ❖ Olasılık veya kullanım sıklıkları bilinen n elemanlı bir küme ile başlanır
- ❖ Semboller azalan sırada olasılıklarına/kullanım sıklıklarına göre sıralanır
- ❖ Semboller kümesi aynı veya yakın olasılık değerlerine göre iki kümeye ayrılır
- ❖ Bir küme içerisindeki sembollerin hepsi **1** değerini alırken,
diğer kümedeki semboller **0** değerini alır
- ❖ Her alt küme kendi içerisinde iki eleman kalıncaya kadar bölünmeye devam edilir
- ❖ Semboller aşağıdan yukarı doğru okunur
- ❖ Sembollerin olasılık değerleri 2'nin negatif kuvvetleri cinsinden olduğu zaman en iyi sonucu verir

sembol
olasılıkları

0,25	1
0,20	1
0,15	0
0,15	0
0,10	0
0,10	0
0,05	0

0,25	1	1
0,20	1	0
0,15	0	1
0,15	0	1
0,10	0	0
0,10	0	0
0,05	0	0

0,25	<u>1</u>	<u>1</u>	
0,20	1	0	
0,15	0	1	1
0,15	0	1	0
0,10	0	0	1
0,10	0	0	0
0,05	0	0	0

kodlar

11
10
011
010
001
0001
0000

$$H = - (0,25 \log_2 0,25 + 0,20 \log_2 0,20 + 2*(0,15 \log_2 0,15) + 2*(0,10 \log_2 0,10) + (0,05 \log_2 0,05)) = 2,67 \text{ bits}$$

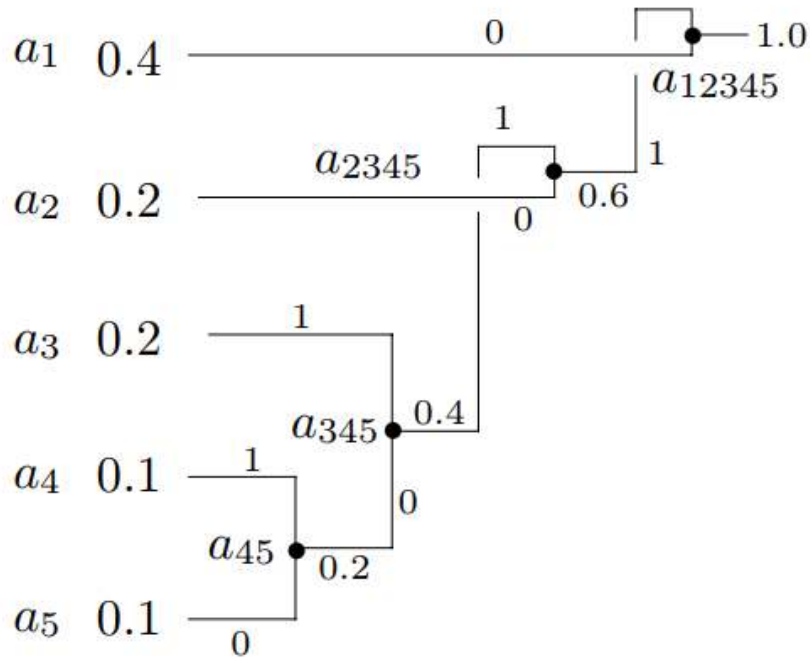
$$S = 0,25 * 2 + 0,20 * 2 + 3* 0,15 * 2 + 3* 0,10 + 4* 0,10 + 4* 0,05 = 2,7 \text{ bits}$$

$$R = |2,70 - 2,67| = 0,03 \text{ bits}$$

0,25	<u>1</u>	<u>1</u>		
0,20	1	0		
0,15	0	1	1	
0,15	0	1	0	
0,10	0	0	1	
0,10	0	0	0	1
0,05	0	0	0	0

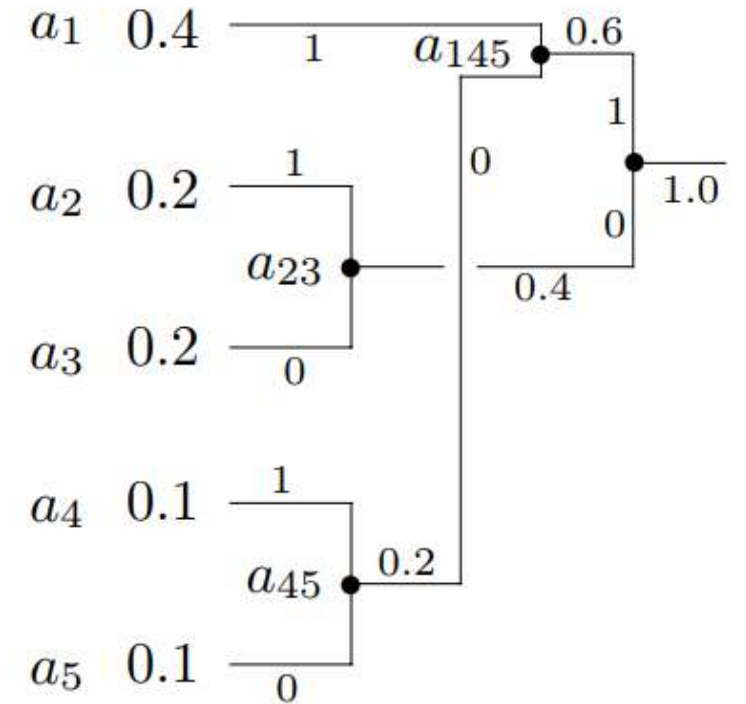
HUFFMAN KODLAMA YÖNTEMİ

- Veri sıkıştırmada tek başına veya bir başka yöntemin içerisinde yer alarak kullanılan bir algoritmadır
- Shannon Fano'ya benzer, ancak kodları ters olarak aşağıdan yukarıya doğru oluşturur
- Huffman algoritmasında kodlama ağacı oluşturulurken iki farklı yol izlenir. Sembollerin kod değerleri farklı olsa da entropy aynı çıkar.



$$S = 0.4 \times 1 + 0.2 \times 2 + 0.2 \times 3 + 0.1 \times 4 + 0.1 \times 4 = 2.2 \text{ bits/symbol}$$

0, 10, 111, 1101, and 1100.



$$S = 0.4 \times 2 + 0.2 \times 2 + 0.2 \times 2 + 0.1 \times 3 + 0.1 \times 3 = 2.2 \text{ bits/symbol}$$

11, 01, 00, 101, and 100.

Hangi kod daha iyidir ?

Birinci ağaç için varyans

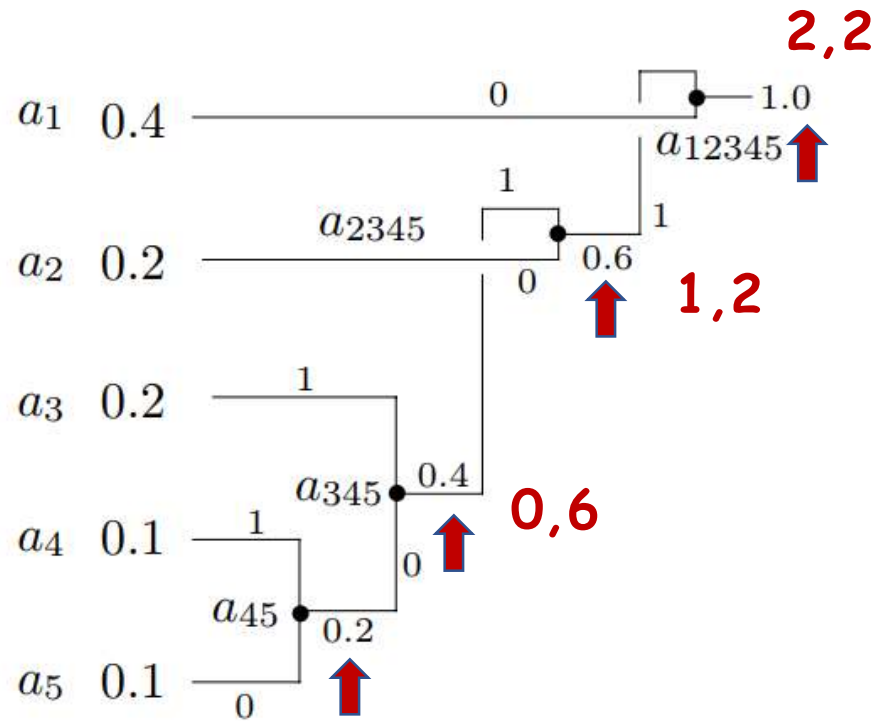
$$0.4(1 - 2.2)^2 + 0.2(2 - 2.2)^2 + 0.2(3 - 2.2)^2 + 0.1(4 - 2.2)^2 + 0.1(4 - 2.2)^2 = 1.36$$

İkinci ağaç için varyans

$$0.4(2 - 2.2)^2 + 0.2(2 - 2.2)^2 + 0.2(2 - 2.2)^2 + 0.1(3 - 2.2)^2 + 0.1(3 - 2.2)^2 = 0.16$$

- Kodlayıcı, sıkıştırılmış veriyi bir dosyaya yazacak ise, kodun varyansının küçük veya büyük olmasının bir önemi yoktur
- Kodlayıcı, sıkıştırılmış veriyi iletişim hatlarından gönderecek ise, küçük varyanslı kod seçilmelidir
- İletişim hattı üzerinden bilginin gönderilme hızı ile, kodlayıcının kodu üretme hızı aynı değilse beklemeyi önlemek için buffer alanı kullanılır.
- Kodun varyansı büyük ise buffer alanı büyük seçilmelidir. Varyansın sıfıra yakın olduğu durumlarda buffer alanı küçük seçilebilir

a_i sembolünün kod uzunluğu $[-\log_2 P_i]$ eşit veya daha küçüktür
 $a_i \rightarrow 0,4 \quad [-\log_2 0,4] = 2 \text{ bits} \quad (2 \text{ veya } 1 \text{ bit ile kodlanır})$



Ortalama kod uzunluğunu hesaplarken, her bir sembolün olasılık değeri ile o sembolün kod bit uzunluklarının çarpımlarının toplamı olarak hesaplanırdı

Ortalama kod uzunluğunu, Huffman kodlama ağacının ara düğüm olasılık değerlerini toplayarak da hesaplayabiliriz

ÖRNEK

Elimizde a, e, k, l, m ve z sembollerinden sırası ile 25, 14, 8, 7, 4 ve 2 adet olsun. Her bir sembol için Huffman kod değerini bulalım.

Adım 1: Sembollerin kullanım sıklıkları küçükten büyüğe doğru sıralanır.

2 4 7 8 14 25

Adım 2: Sıklık değerleri, soldan başlayarak sıradaki iki sıklık değerinin toplamalarının sıralamayı bozmayacak şekilde diziye yerleştirilmesi ile devam edilir.

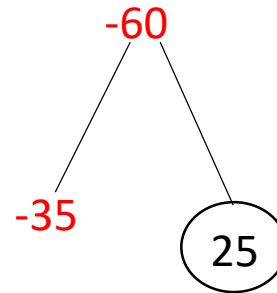
2 4 -6 7 8 -13 14 -21 25 -35 -60

Toplam değerleri kırmızı ile gösterilmiştir. Eksi değer de bir ara toplam olduğunu söyler. n tane sembolümüz varsa, genişletilmiş dizimiz $2n-1$ adet olacaktır.

2 4 -6 7 8 -13 14 -21 25 -35 -60

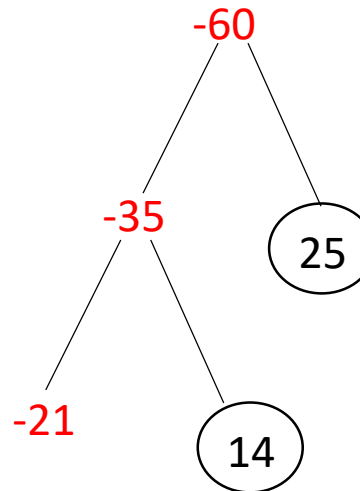
Adım 3: En sondaki eleman Huffman kodlama ağacının kök değeri olacak şekilde ikili ağaç şeklinde diğer elemanlarda ağaca belli bir kural ile yerleştirilir.

Bir sonraki $(2n-2)$ değeri ağacın sol dalına, $(2n-3)$ değeri de sağ dalına yerleşir.



25 değeri "a" sembolün sıklık değeri olup, bir yapraktır.

-35 ise bir ara düğümdür. Ağaç bu düğüm üzerinden oluşturulmaya devam edilir.



2

4

-6

7

8

-13

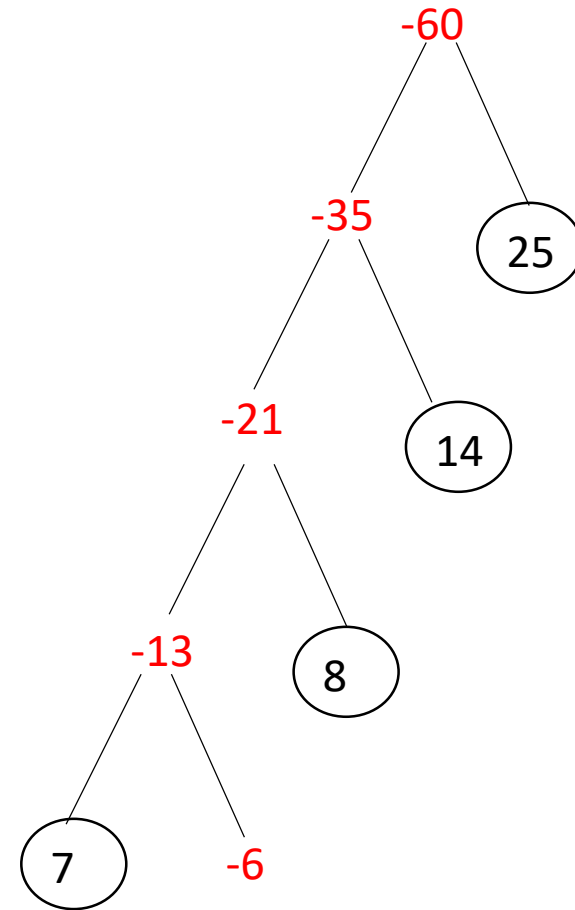
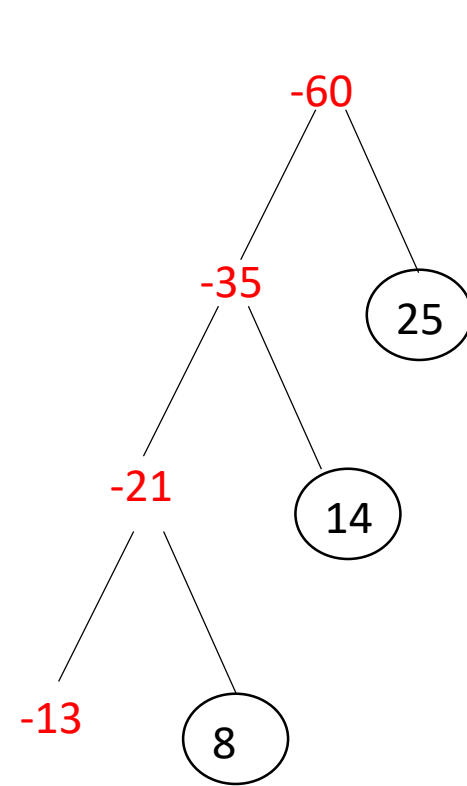
14

-21

25

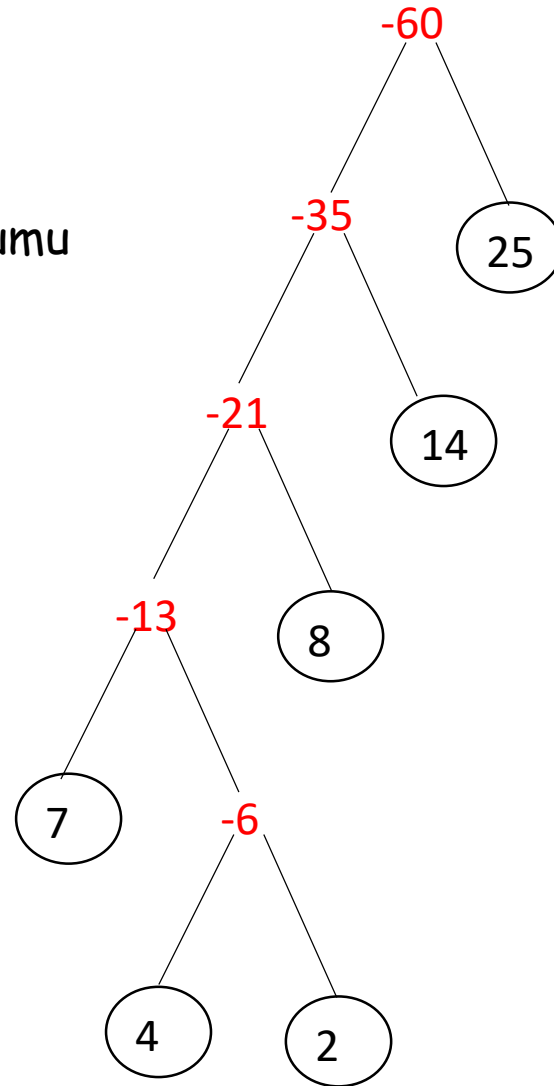
-35

-60

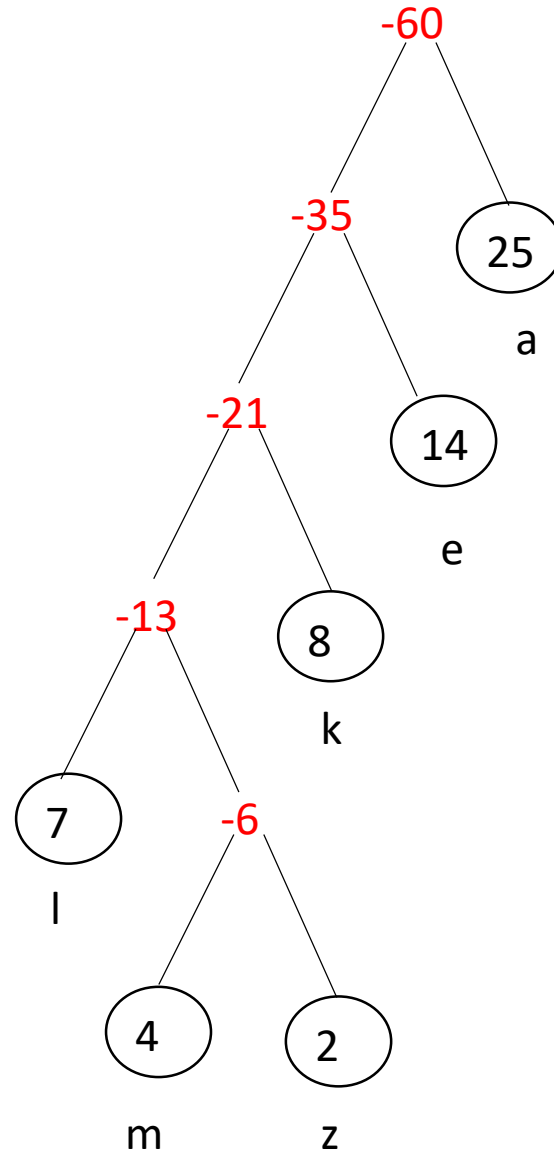


2 4 -6 7 8 -13 14 -21 25 -35 -60

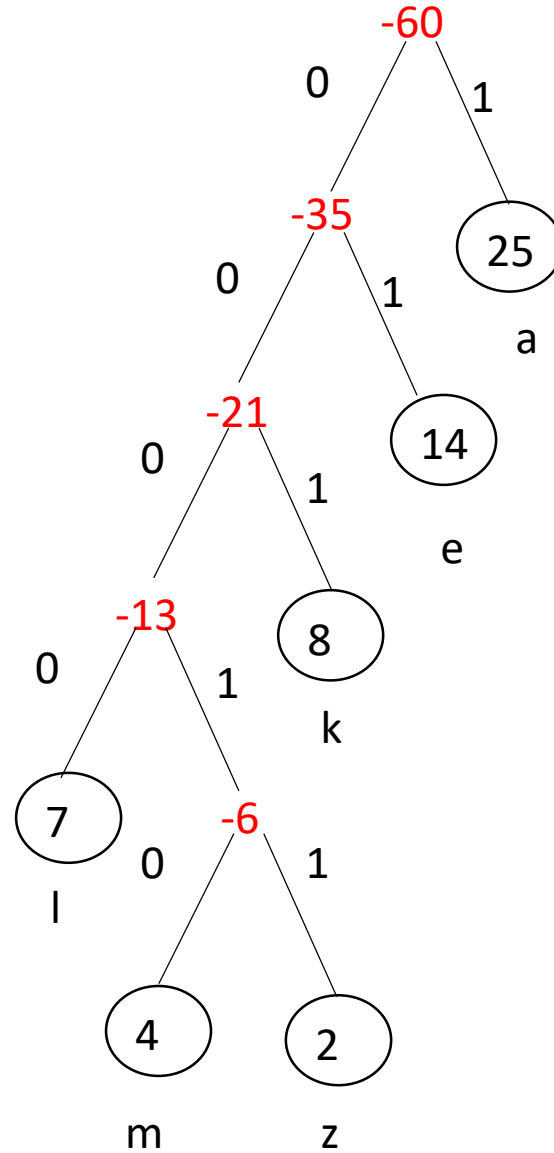
Huffman Kodlama Ağacının son durumu



Huffman Kodlama Ağacının yapraklarına sembolleri yerleştirelim.



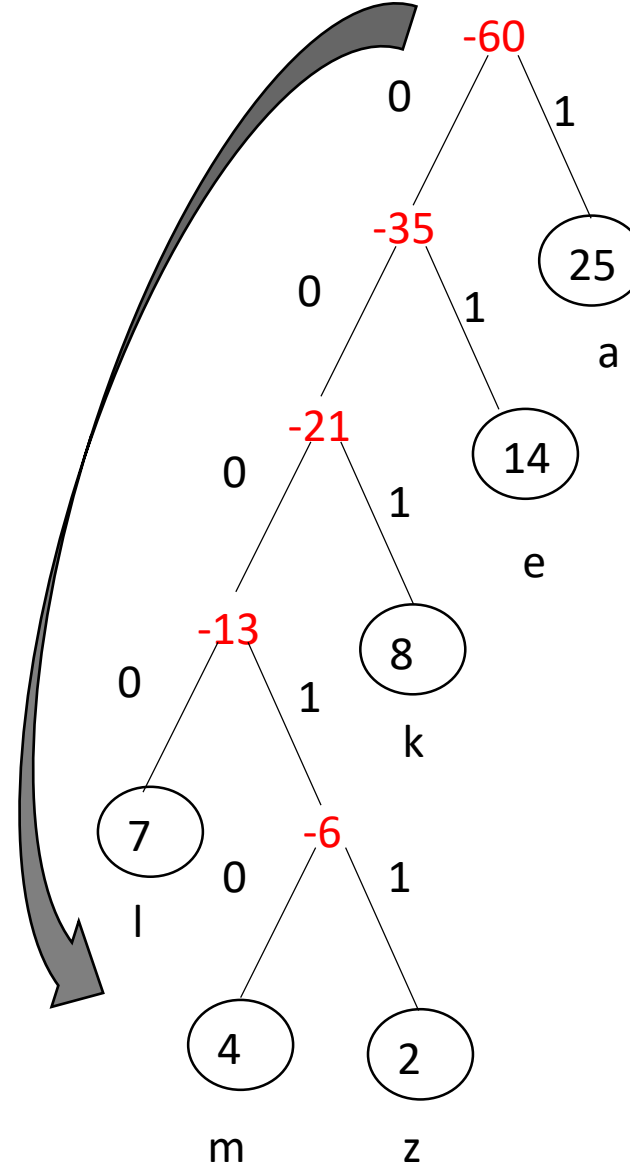
Adım 4: Ağacın dallarına Huffman kodlarını oluşturacak olan 1 ve 0 değerlerini yerleştirelim. Sol dal için 0, sağ dal için 1 değerini yazalım.



Adım 5: Huffman kodlama ağacında kodlar aşağıdan yukarı doğru oluşturulur, yukarıdan aşağı doğru okunur.

Ağaç üzerinden kodlarımızı okursak

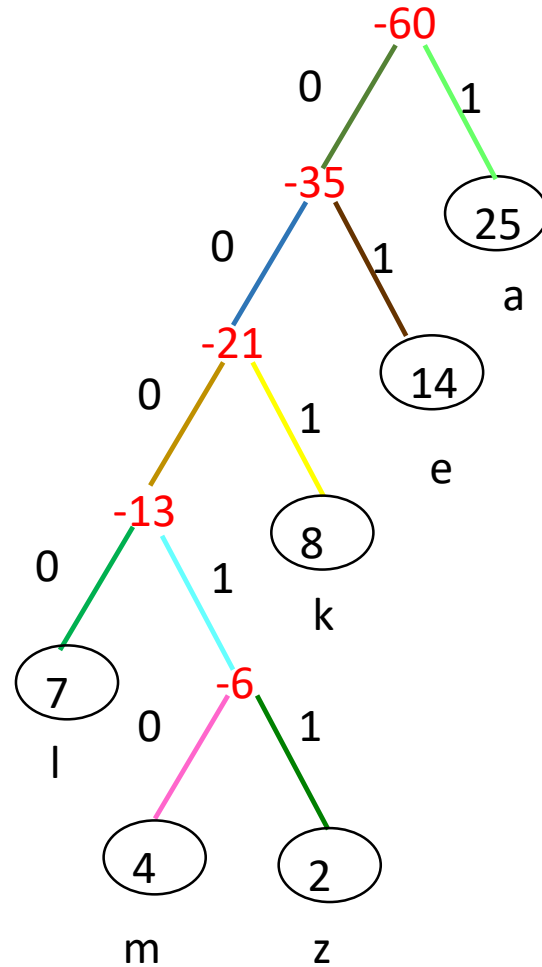
a	1
e	01
k	001
l	0000
m	00010
z	00011



Header (Başlık Bilgisi): Başlık içerisinde ağaç bilgisini gönderebilir miyiz?

Nasıl?

Ağacımızı Left Node Right notasyonuna göre okuyalım



0000101111

lmzkea

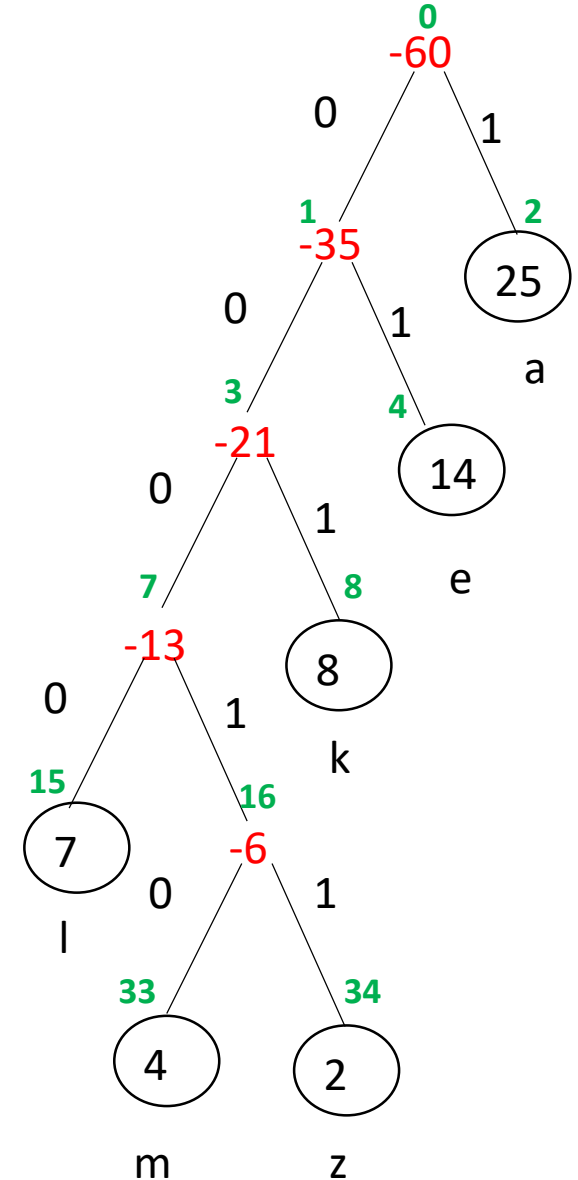
Genişletilmiş diziye kullanarak kodları nasıl oluşturabiliriz ?

2 4 -6 7 8 -13 14 -21 25 -35 -60

Ağaç üzerindeki her düğüm noktasının adreslerini yazalım

Kök düğümün adresi 0 ile başlasın (yeşil ile yazılmış sayılar)

- sol tarafındaki düğüm kendisinin $*2 + 1$ olsun
- sağ taraftaki düğüm de $*2 + 2$ olsun

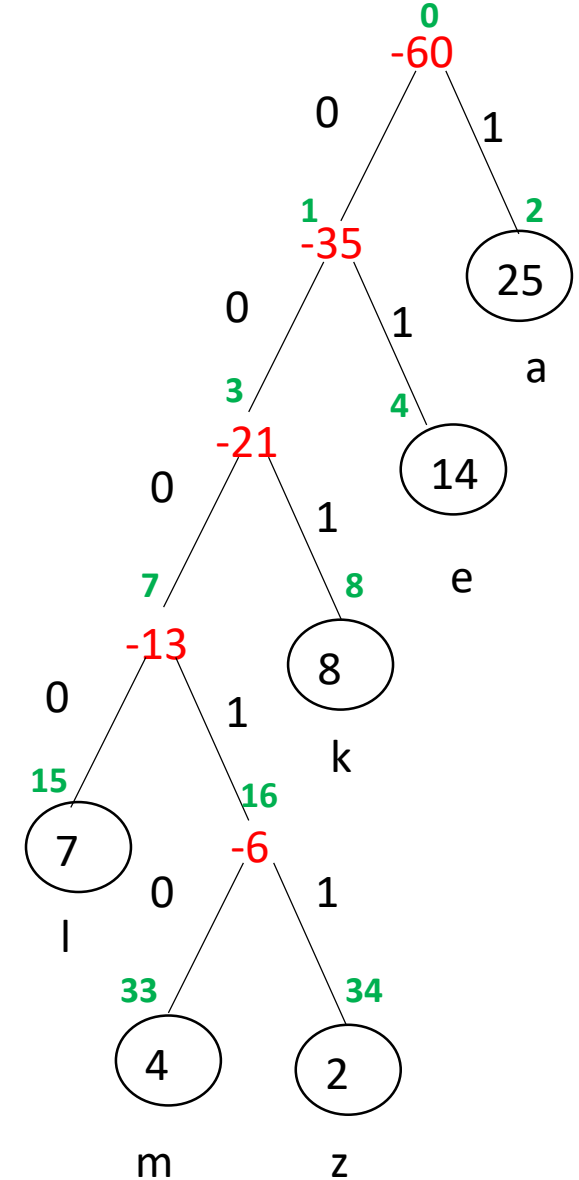


2 4 -6 7 8 -13 14 -21 25 -35 -60

Satır sayısı $2n-1$, sütun sayısı 3 olan bir matris hazırlayalım

- İlk sütun genişletilmiş dizi elemanları (tersten)
- İkinci sütun indislerden
- Üçüncü sütun da kodlardan oluşsun

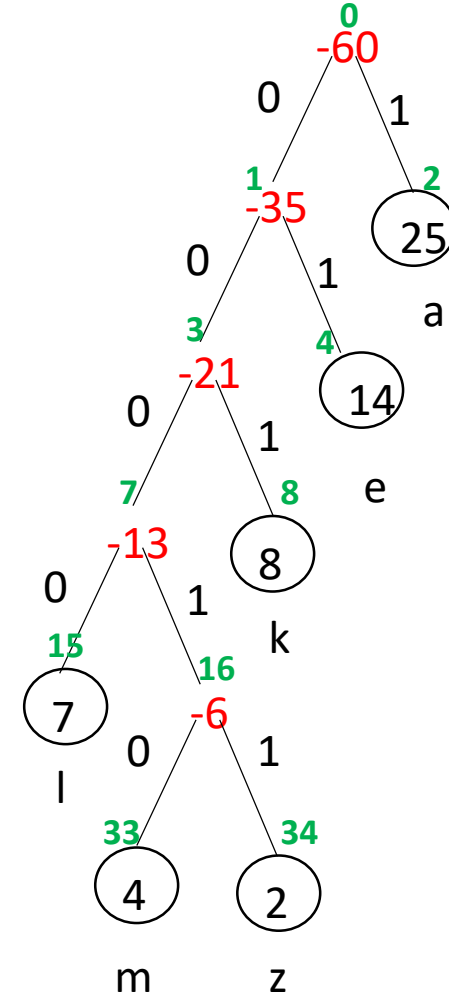
dizi	indis	kod
-60	0	
-35		
25		
-21		
14		
-13		
8		
7		
-6		
4		
2		



Her negatif sayı bir ara düğüm olduğu için bu düğümden çıkan dallar var demektir.
Bu sebeble her negatif sayının sahip olduğu indis değerinin $*2+1$ ile $*2+2$ değeri peş peşe indis sütununa yazılır.

Örneğin: -60 bir ara düğümdür. Indis değeri 0 dır. $*2+1$ ve $*2+2$ değerleri sırası ile 1 ve 2 olup, Arka arkaya yazılır.

dizi	indis	kod
-60	0	
-35	1	
25	2	
-21	3	
14	4	
-13	7	
8	8	
7	15	
-6	16	
4	33	
2	34	



- Huffman Ağacı kodları aşağıdan yukarı doğru oluşturduğundan kodların prefix özelliğini sağlaması içinde yukarıdan aşağıya doğru okunması gerekir.
- Matriste kod sütunun oluşturulması sırasında da ağaca indisleri yerleştirirken yaptığımız işlemin tersini yapmamız gerekir.

Dizinin pozitif elemanları ağacın yapraklarına karşılık gelip, sembolü temsil ettiğinden sahip olduğu indis değeri ile işlem yapılır, elde edilen kod değerleri de sağdan sola doğru yazılır:

```

While indis ≠ 0 do
{
    if indis çift then
    {
        print "1"
        indis = (indis-2)/2 }
    else
    {
        print "0"
        indis = (indis-1)/2 }
}

```

dizi	indis	kod
-60	0	
-35	1	
25	2	1
-21	3	
14	4	01
-13	7	
8	8	001
7	15	0000
-6	16	
4	33	00010
2	34	00011

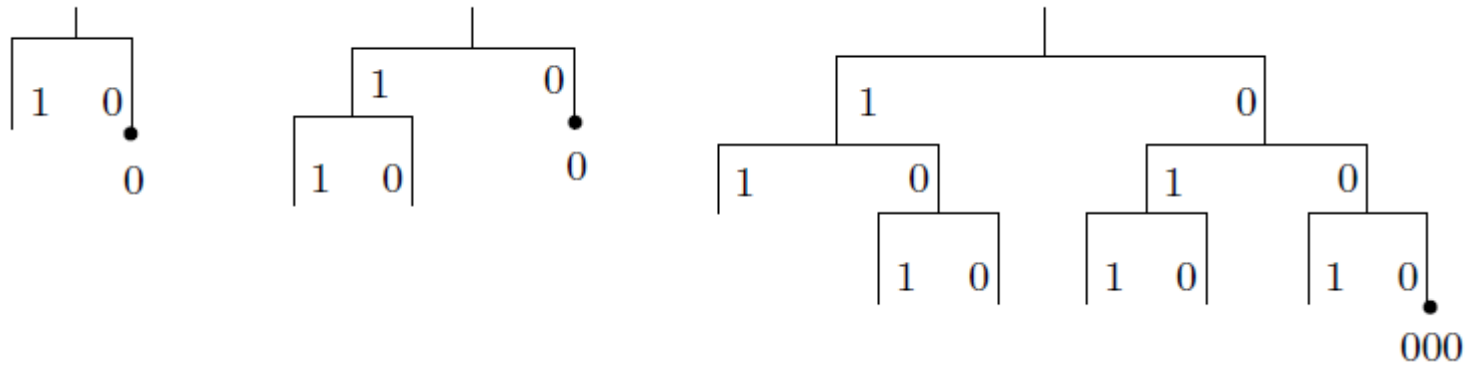
Statik Huffman Kod : Bir derlem içerisinde yer alan tüm sembollerin kullanım sıkları çıkarıldıktan sonra, her sembol için Huffman kod değerleri oluşturulur. Gelen her doküman için aynı kod değerleri kullanılır. Doküman içerisindeki sembollerin kullanım sıkları dikkate alınmaz.

Yarı Dinamik Huffman Kod : Kodlanacak her doküman üzerinde iki aşamadan oluşan bir işlem yapılır. İlk olarak doküman baştan sona taranarak sembollerin kullanım sıklıkları çıkarılır. İkinci adımda da Huffman kod değerleri elde edilir.

Dinamik Huffman Kod : Hızın önemli olduğu uygulamalarda kullanılır.

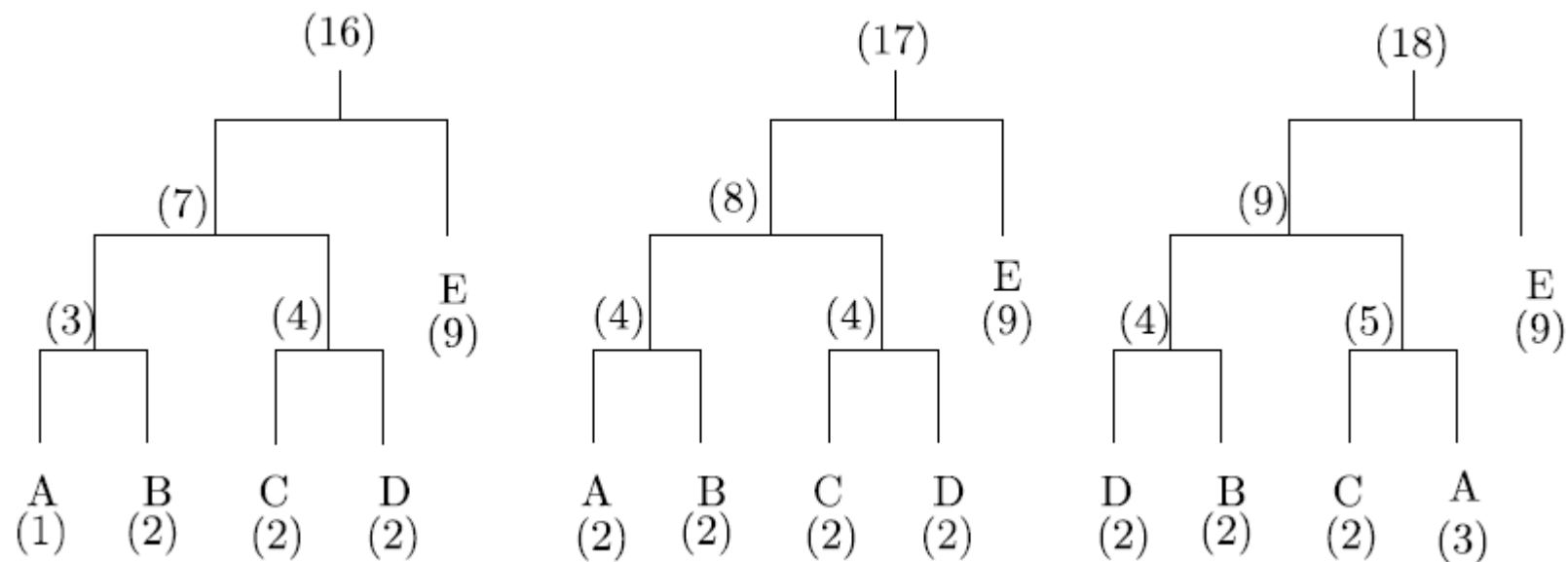
- ✓ Encoder ve Decoder senkronize çalışarak aynı anda kodlama ağacını oluşturur.
- ✓ Başlangıçta ağaç boştur. Okunan her yeni sembol ile ağaç yeniden oluşturulur.
- ✓ Input stream den ilk sembol okunur ve kodlanmadan output stream yazılır.
- ✓ Sembol ağaca eklenir ve kendisine bir kod atanır.
- ✓ Daha sonraki adımlarda bu sembol ile karşılaşırsa, önce sembolün kodu gönderilir, sonra da frekans değeri bir artırılır.
- ✓ Decoder'ın bilmesi gereken, gönderilen sembolün kodlanarak mı yoksa kodlanmadan mı gönderildiğidir. Kodlanmamış sembolden önce escape kodu gönderilir.
- ✓ Decoder, escape kodundan sonra gelen kodun ASCII koda sahip bir sembol olduğunu anlar.
- ✓ Escape kodun frekans değeri her zaman 1 dir ve değişmez. Ancak, her yeni sembol ağaca eklendikçe kodu değişir.

Escape kodu ve ağaç üzerindeki değişimi

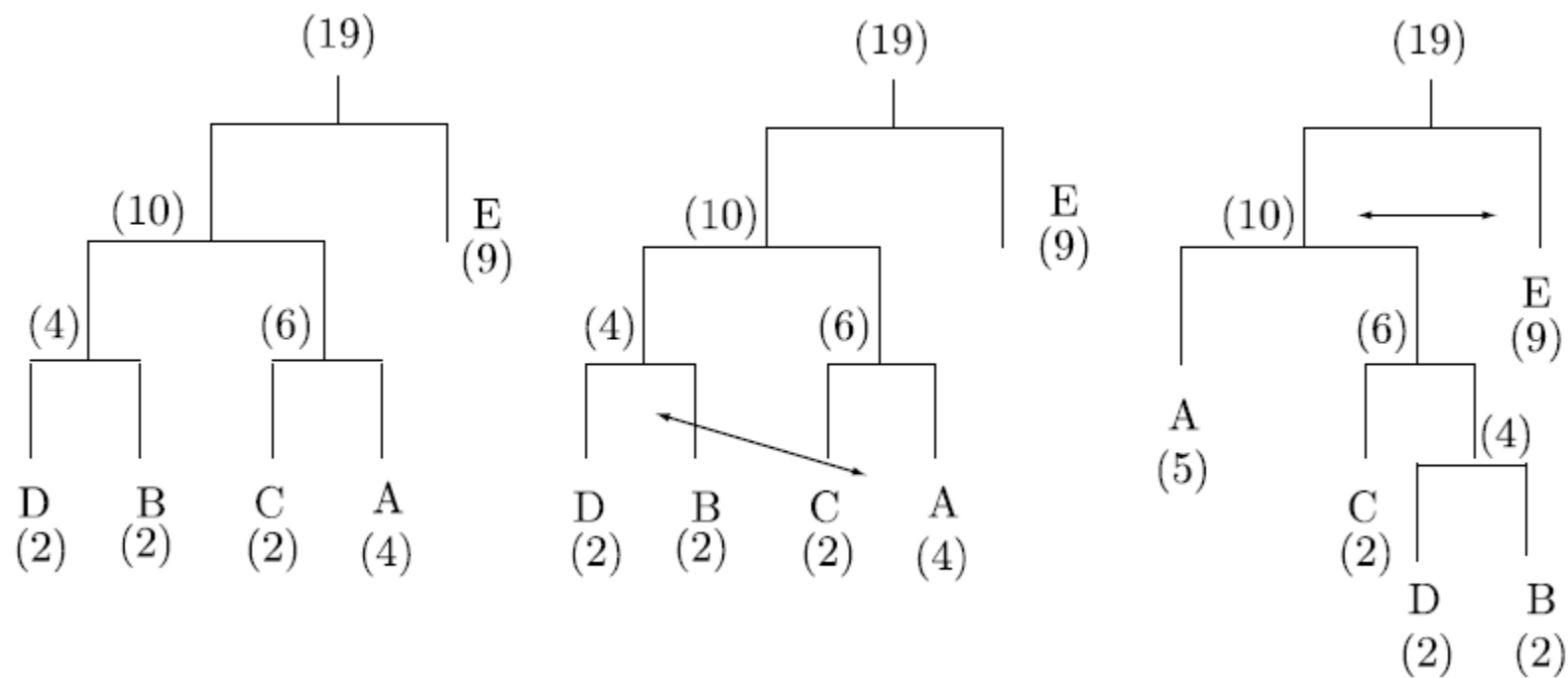


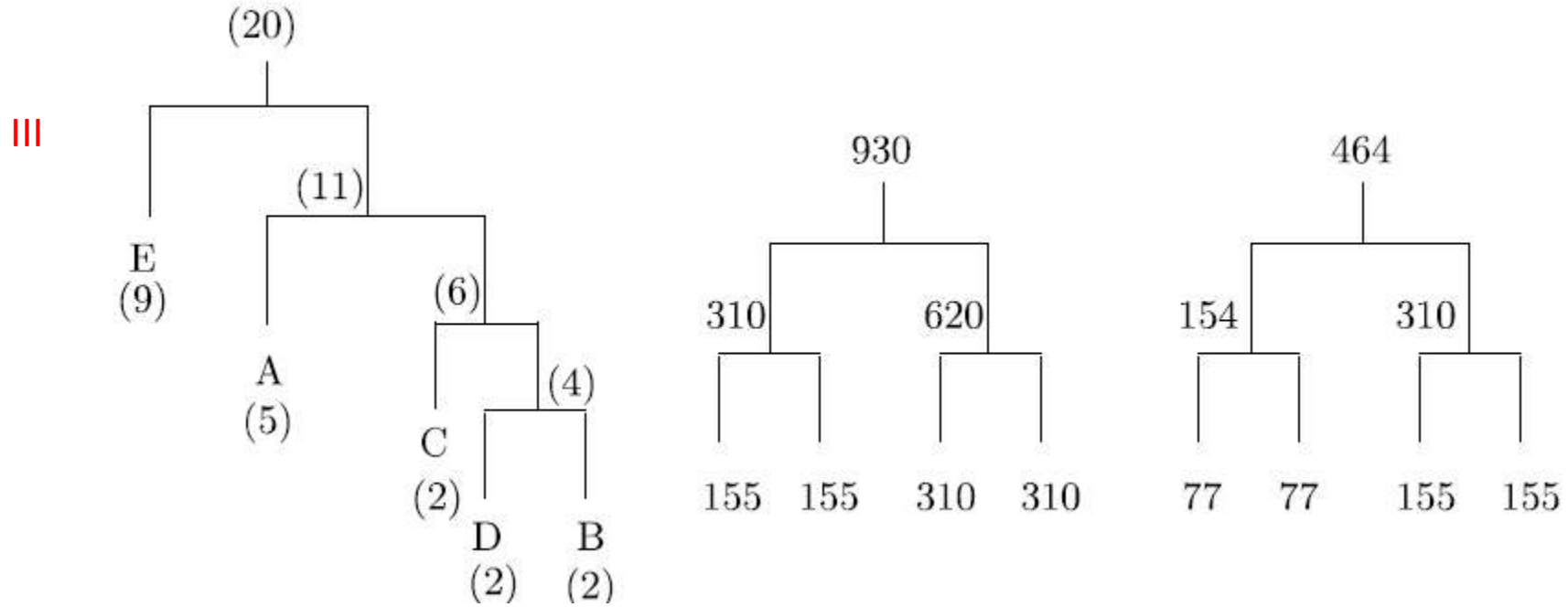
Dinamik olarak Huffman ağacı nasıl oluşturuluyor ?

I



II





Bu algoritmada aksayan taraflar nelerdir ?

Sayıcının taşması

Frekans değerinin tutulduğu olan işaretli 16 bit olsun. Bu değışkende ($2^{16}-1=65535$) değ tutabiliriz. Taşmanın oluşmaması için kökteki frekans değeri kontrol edilir. Maksimum sayı değeri ulaşıldığında yeniden ölçeklendirme yapılır. Yaprakların değeri yarısı alınır (integer bölme). Ara düğümler ve kök tekrardan hesaplanır.

Kod taşması

Kodların tutulduğu alan 16 bit ve ağacın seviyesi de 16'dan fazla ise ya bu alan genişletilir ya da linkli liste kullanılır.

Huffman Algoritmasının Farklı Bir Versiyonu

- Basit, ancak daha az etkili bir yöntemdir
- Eşit olasılıklı, n elemanlı değişken uzunluktaki kodlar için uygundur
- Sembollerin olasılıkları eşit değilse etkili değildir
- $n \times 3$ boyutunda bir tablodan oluşup, kolon değerleri sembolün kendisi, sıklığı ve kodudur
- İkinci kolonun değeri değiştikçe satırlar yer değiştirir ancak üçüncü kolon sabit kalır

Name	Count	Code	Name	Count	Code	Name	Count	Code	Name	Count	Code
a_1	0	0	a_2	1	0	a_2	1	0	a_4	2	0
a_2	0	10	a_1	0	10	a_4	1	10	a_2	1	10
a_3	0	110	a_3	0	110	a_3	0	110	a_3	0	110
a_4	0	111	a_4	0	111	a_1	0	111	a_1	0	111

Bu yöntemde count değerinde taşma oluşur, bunu çözmek için count değeri yarıya bölünür

Canonical Huffman Kodlama

- Huffman prefix kodları ile sıkıştırılmış dosyaların başlıklarına kıyasla, geniş alfabeli dosyalarda daha küçük dosya başlıkları oluşturmaya olanak sağlar
- Hızlı kod dönüşüme sahiptir
- Huffman algoritması ile oluşturulamayıp atlanan prefix kodların oluşturulmasına izin verir

Nasıl Oluşturulur ?

Canonical Huffman kod oluşturma işleminin ilk aşaması Huffman kodlarının oluşturulmasıdır. Alfabedeki her bir karakterin sahip olduğu Huffman kod uzunlukları kullanılarak Canonical Huffman kod oluşturulur.

Ne işe yarıyor ?

Canonical Huffman kod yöntemiyle kodlanmış veriler için küçük bir başlık alanına ihtiyaç duyulması ve kodların hızlı geri dönüşüm özelliklerinden dolayı, hızlı kod çözme gerektiren uygulama alanlarında kullanılmasına olanak sağlar (sıkıştırılmış ses, görüntü ve arşiv uygulamaları)

Canonical Huffman kod nasıl oluşturulur ?

- Kodlanacak alfabede yer alan her bir karakter için Huffman prefix kod değeri geleneksel Huffman yöntemi ile oluşturulur
- Prefix kod uzunlukları büyükten küçüğe doğru sıralanır
- En uzun prefix kodun değeri L_{\max} ve bu uzunlukla k_1 tane prefix kod varsa, L_{\max} uzunluğunda 0'dan başlamak üzere k_1 tane ardışık binary sayı ile yeni Canonical Huffman prefix kodu üretilir
- L_{\max} 'dan küçük olan bir sonraki uzunluk değeri $L_{\max-1}$ ve bu uzunlukta k_2 tane prefix kod var ise, bir önceki prefix kod değerinin ilk $L_{\max-1}$ biti alınır ve k_2 tane yeni Canonical Huffman prefix kodunun herbiri için 1 eklenir
- Bir önceki adımda yapılan işlemler, büyükten küçüğe doğru tüm uzunluk değerleri için tamamlanır

Örnek

Huffman algoritması kullanılarak {a, b, c, d, e, f, g ve h} alfabeti için elde edilen Huffman kodlarının uzunluk değerleri sırası ile (5, 5, 5, 5, 3, 2, 2, 2) verilmiş olsun. Bu alfabe için hesaplanacak olan Canonical Huffman kodları ne olur ?

$$L_{\max} = 5 \quad k_1 = 4$$

```
00000
00001
00010
00011
```

$$L_{\max-1} = 3 \quad k_2 = 1$$

00011

```
  000
+   1
-----
  001
```

$$L_{\max-2} = 2 \quad k_3 = 3$$

001

00	01	10
+ 1	+ 1	+ 1
-----	-----	-----
01	10	11