

# Görüntü İşleme 3



# İmge Sıkıştırma

*Image Compression*

# İmge Sıkıştırma

İmgeyi en az veri ile ifade etmek için kullanılır. İmge;

- Bellekte daha az yer kaplar
- Aynı bellekte daha fazla veri saklanmasını sağlar
- İletimi kolaylaştırır (Bant genişliğini azaltır)
- Yazılım ya da donanım yükünü arttırır

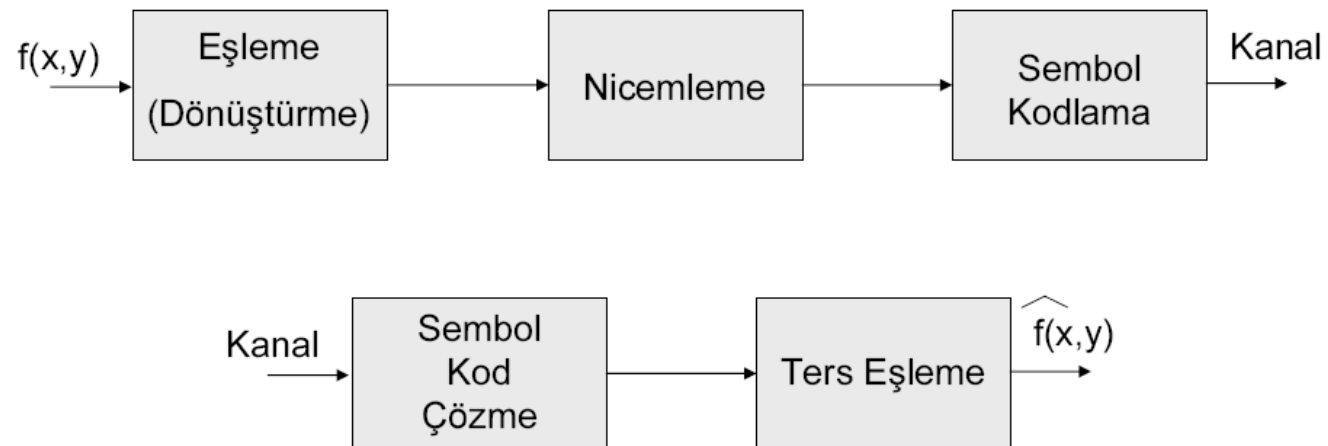
# İmge Sıkıştırma

Genel sıkıştırma sisteminin yapısı:



# İmge Sıkıştırma

Genel sıkıştırma sisteminin yapısı:



## İmge Sıkıştırma

Eşleyici: Pikseller arası artıklığı sıkıştırma için uygun biçime dönüştürme işlemi yapar. Imgenin yeniden elde edilebilmesi için tersi alınabilen dönüşümler kullanılır.

Nicemleyici: Eşleyicinin çıkışındaki verinin verimliliğini, önceden belirlenmiş belirli kurallar çerçevesinde düşürür. Geri dönüşümü yoktur. Bu nedenle kayıpsız sıkıştırmada ihmal edilmelidir.

Sembol Kodlayıcı: Nicemleyici çıkışındaki veriyi sabit veya değişken uzunluklu kodlar.

# İmge Sıkıştırma-JPEG

JPEG (Joint Photographic Experts Group ):

- 1992 yılında uluslararası standart olarak belirlendi.
- Kayıplı bir sıkıştırma tekniğidir.

-----

➤ Kayıpsız sıkıştırma yöntemlerinin veri sıkıştırma oranları düşüktür (Huffmann, LZW, Aritmetik kodlama...). JPEG'de ise yüksek sıkıştırma oranları elde edilebilir.

# İmge Sıkıştırma- JPEG

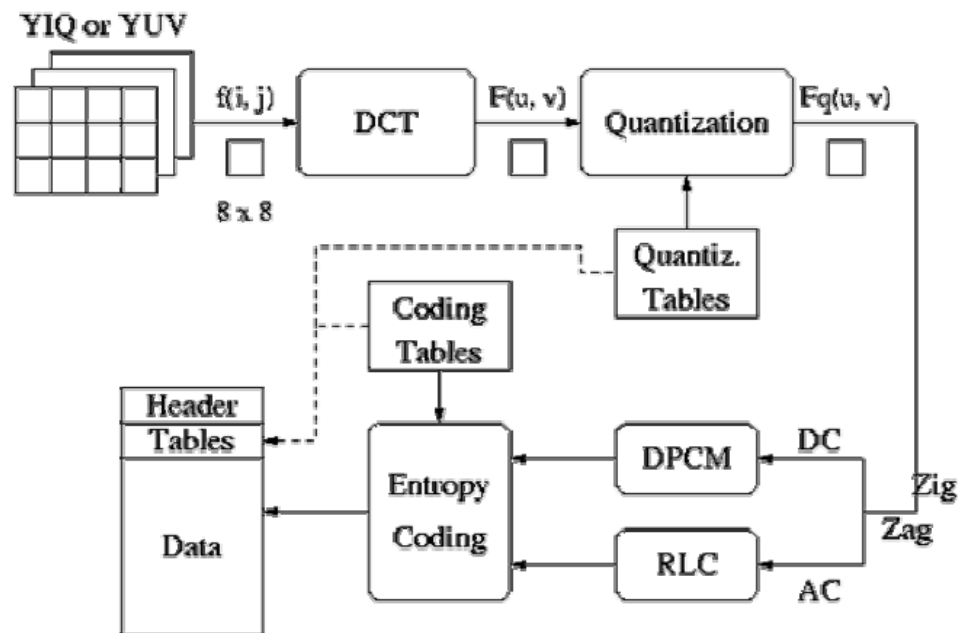
## JPEG:

- Dönüşüm kodlama kullanır.
- Düşük uzamsal frekans bileşenleri, yüksek frekans bileşenlerinden daima daha fazla bilgi taşır (yüksek frekans bileşenlerinde gereksiz detaylar ve gürültü bilgileri de bulunur).



# İmge Sıkıştırma- JPEG

## Kodlayıcı Yapısı:



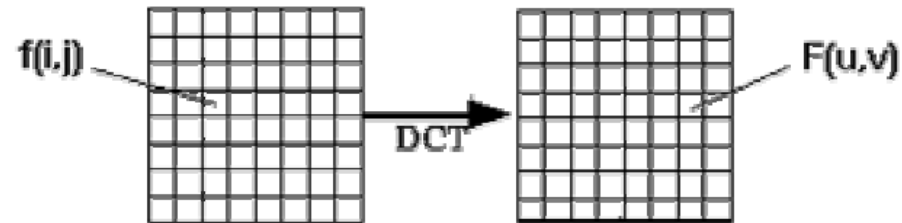
## İmge Sıkıştırma- JPEG

JPEG'de blokların 8x8 olmasının nedeni, tümdevre teknolojisi ile kullanılmasının daha kolay olmasındandır. Tek bir işlem süresinde alınan bir blok kolaylıkla işlenebilir.

JPEG' de ayrık kosinüs dönüşümü (Discrete Cosine Transform-DCT) kullanılmaktadır. Fourier dönüşümünde sinüs ve kosinüs bileşenleri kullanılırken, ayrık kosinüs dönüşümünde adından da anlaşılacağı üzere yalnızca kosinüs bileşenleri kullanılmaktadır. Yani, işaret kosinüs bileşenleri ile ifade edilmektedir.

# İmge Sıkıştırma- JPEG

Ayrık Kosinüs Dönüşümü (DCT):



$$F(u, v) = \frac{\Delta(u)\Delta(v)}{4} \sum_{i=0}^7 \sum_{j=0}^7 \cos \frac{(2i+1) \cdot u\pi}{16} \cdot \cos \frac{(2j+1) \cdot v\pi}{16} \cdot f(i, j)$$
$$\Delta(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

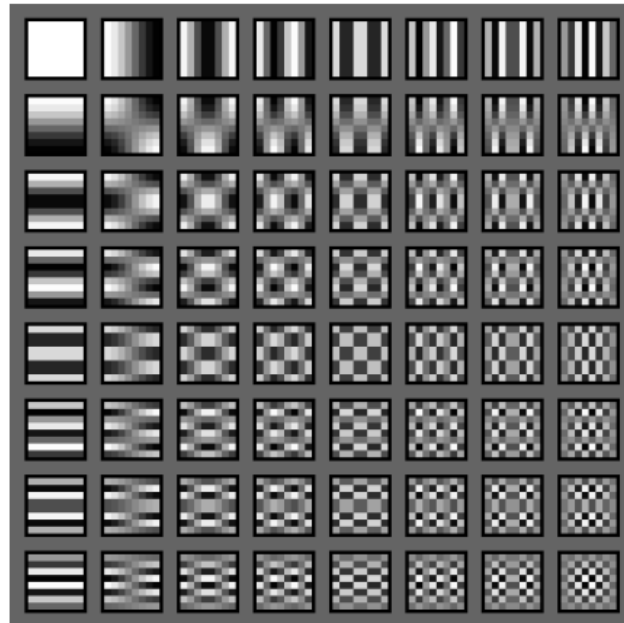
# İmge Sıkıştırma- JPEG

Ayrık Ters Kosinüs Dönüşümü (IDCT):

$$\hat{f}(i,j) = \frac{1}{4} \sum_{u=0}^7 \sum_{v=0}^7 A(u)A(v) \cos \frac{(2i+1) \cdot u\pi}{16} \cdot \cos \frac{(2j+1) \cdot v\pi}{16} \cdot F(u,v)$$
$$A(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

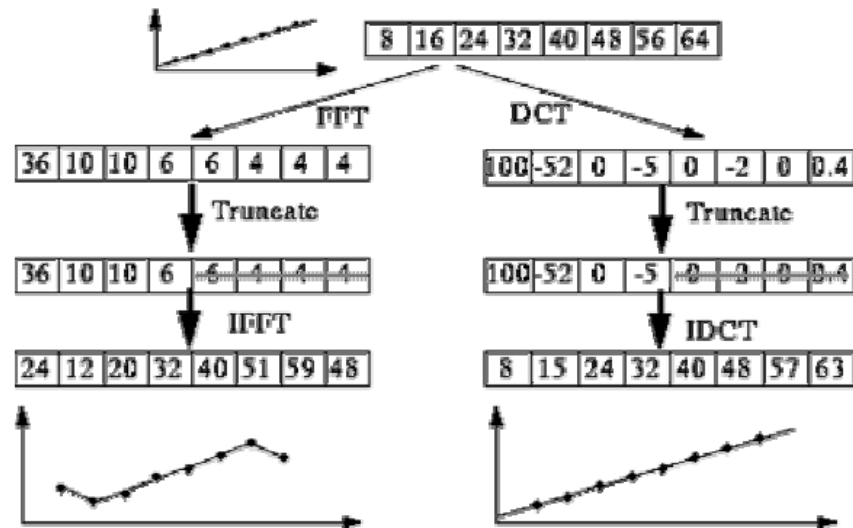
# İmge Sıkıştırma- JPEG

8x8 DCT Temel Fonksiyonları:



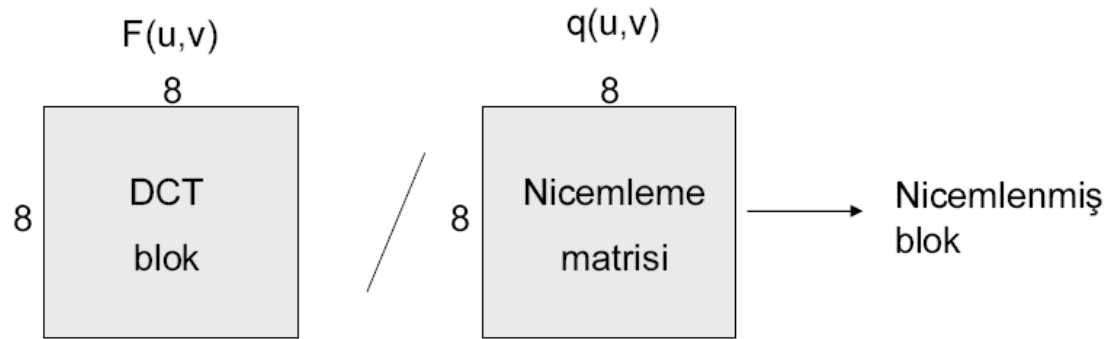
# İmge Sıkıştırma- JPEG

Neden FFT değil de DCT kullanılıyor !!!



# İmge Sıkıştırma- JPEG

Nicemleme:



$$F'[u, v] = \text{round} ( F[u, v] / q[u, v] ).$$

Bu işlemin yapılmasının nedeni ne olabilir?

## İmge Sıkıştırma- JPEG

Örnek:  $101101 = 45$  (6 bit).

$q[u, v] = 4 \rightarrow 4$  bit'e sınırlandır :  $1011 = 11$ .

**SONUÇ:** Kayıplı sıkıştırmanın temel nedeni nicemleme hatasıdır.



# İmge Sıkıştırma- JPEG

Nicemleme:

- Düzgün dağılımlı nicemleme:

Her  $F(u,v)$  değeri sabit bir sayıya bölünür

- Düzgün olmayan dağılımlı nicemleme:

Özel nicemleme matrisi oluşturulur

1	1	1	1	1	2	2	4
1	1	1	1	1	2	2	4
1	1	1	1	2	2	2	4
1	1	1	1	2	2	4	8
1	1	2	2	2	2	4	8
2	2	2	2	2	4	8	8
2	2	2	4	4	8	8	16
4	4	4	4	8	8	16	16

Düşük sıkıştırma oranı,  
yüksek kalite

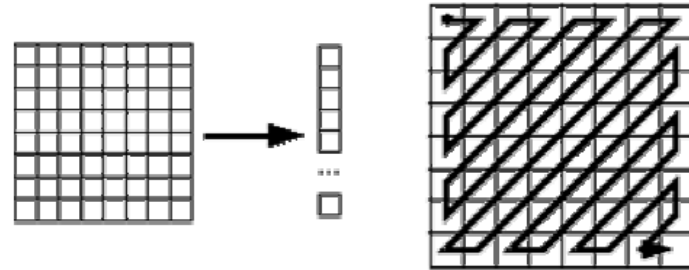
1	2	4	8	16	32	64	128
2	4	4	8	16	32	64	128
4	4	8	16	32	64	128	128
8	8	16	32	64	128	128	256
16	16	32	64	128	128	256	256
32	32	64	128	128	256	256	256
64	64	128	128	256	256	256	256
128	128	128	256	256	256	256	256

Yüksek sıkıştırma oranı,  
düşük kalite

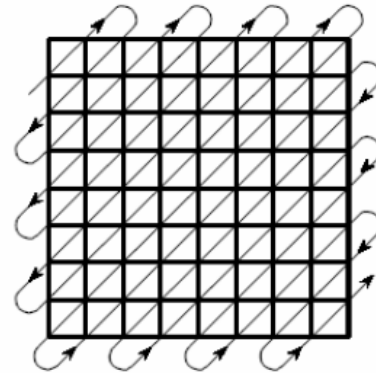
# İmge Sıkıştırma- JPEG

Zig Zag Tarama:

Düşük frekans katsayılarını sırayla almak için kullanılır.



Sonuçta; 1x64'lük dizi oluşur.



# İmge Sıkıştırma- JPEG

Farksal Darbe Kod Modülasyonu

(Differential Phase Code Modulation-DPCM):

Blokların DC katsayıları büyük ve değişkendir fakat yanındaki bloğun DC değerine yakındır.

Bu nedenle, DPCM ile DC değerlerin farkı kodlanır.

# İmge Sıkıştırma- JPEG

Dizi Uzunluğu Kodlaması

(Run Length Encoding-RLE):

Zig zag tarama sonrası oluşan 1x64'lük dizinin içerisinde çok sayıda "0" vardır.

Bu sıfırlar yüksek frekans bileşenlerini içermekteydi.

RLE ile [...1 1 0 0 0 0 0 2 ...]

[..... 6 0.....]

şeklinde kodlanır.

## İmge Sıkıştırma- JPEG

RLE ile kodlanmış dizi en son Entropi Kodlama'dan geçirilerek JPEG dosyası oluşturulur.

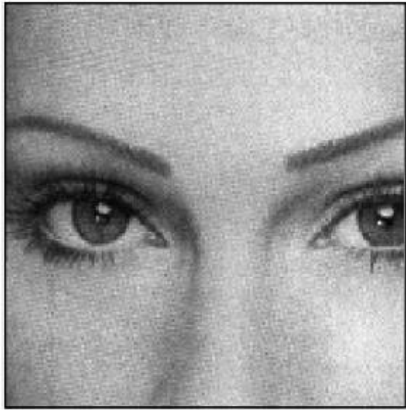
Entropi kodlama olarak:

Aritmetik Kodlama

Huffman Kodlama

kullanılır.

## İmge Sıkıştırma-PSNR Hesabı



orijinal imge



10:1 sıkıştırma oranı



45:1 sıkıştırma oranı

## İmge Sıkıştırma-PSNR Hesabı

Tepe İşaret/Gürültü Oranı

(Peak Signal to Noise Ratio-PSNR):

Orijinal imge ile sıkıştırılmış ya da farklı işlemlerden geçirilmiş imgenin benzerlik ölçütünü verir.

$$MSE = \frac{1}{MN} \sum_{y=1}^M \sum_{x=1}^N [I(x,y) - I'(x,y)]^2$$

$$PSNR = 20 * \log_{10} (255 / \sqrt{MSE})$$

$I(x,y)$ : orijinal imge

$I'(x,y)$ : değiştirilmiş imge

# Huffman Kodlaması



Huffman kodlaması, kayıpsız sıkıştırma algoritmasıdır.



Bu kodlama değişken uzunlukta bir kodlama olup bu kodlama mors kodlamasının tersine veri setindeki karakterlerin frekansına bağlıdır.



Algoritma, bir veri setinde daha çok rastlanan bir sembolü daha düşük uzunluktaki kodla, daha az rastlanan sembolü ise daha büyük uzunluktaki kodla temsil edilmesine dayanmaktadır.



Veri setindeki sembol sayısına ve bu sembollerin tekrarlama sayısına bağlı olarak %10 ile %90 arasında değişen oranlarda bir sıkıştırma elde edilebilir.



## Huffman Kodlaması (Devam)



Huffman tekniğinde semboller(karakterler) ASCII'de olduğu gibi sabit uzunluktaki kodlarla kodlanmazlar. Her bir sembol değişken sayıda uzunluktaki kod ile kodlanır.



Bir veri kümesini sıkıştırabilmek için bu küme içerisindeki sembollerin tekrar etme sıklıklarının bilinmesi gerekmektedir. Her sembolün ne kadar sıklıkta tekrar ettiğini gösteren tabloya frekans tablosu denir.

## Huffman Kodlaması (İşlem Adımları)



İlk olarak, veri setine ait frekans tablosu oluşturulur



Ardından, hangi karakterin hangi bitlerle temsil edileceğini gösteren Huffman ağacı oluşturulur.

## Örnek 1

Veri setine ait frekans tablosu aşağıdaki gibi olsun.

Sembol	Frekans
A	60
B	40
C	25
D	20
E	10

## Adım 1



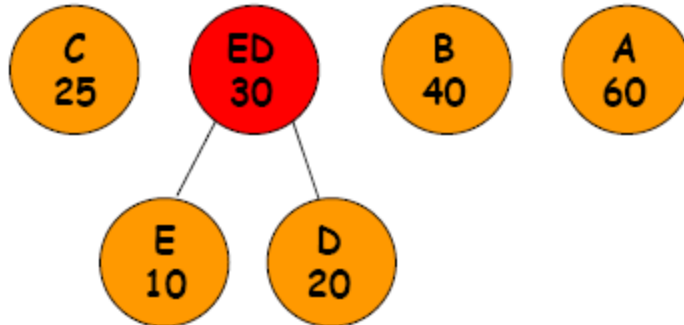
Huffman ağacındaki en son düğümleri oluşturacak olan bütün semboller frekanslarına göre küçükten büyüğe doğru sıralanırlar.



## Adım 2



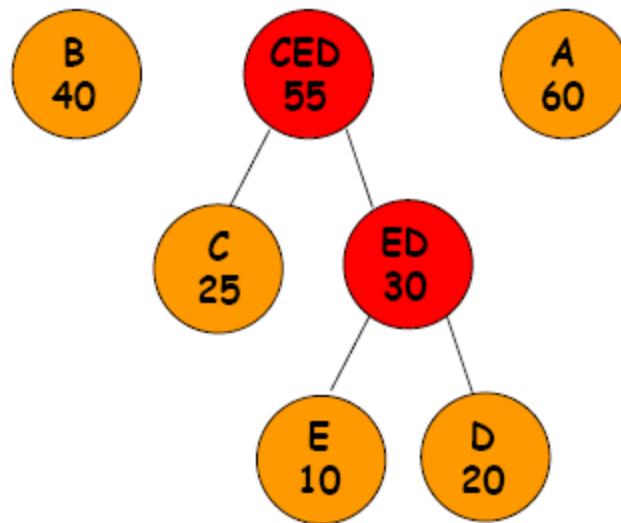
En küçük frekansa sahip olan iki sembolün frekansları toplanarak yeni bir düğüm oluşturulur. Oluşturulan bu yeni düğüm var olan düğümler arasında uygun yere yerleştirilir. Bu yerleştirme frekans bakımından küçüklük veya büyüklüğe göredir.



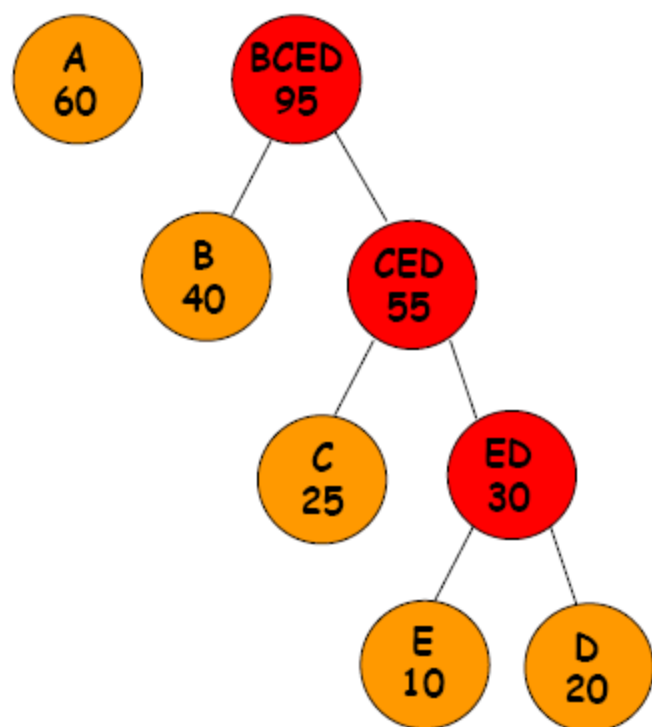
## Adım 3



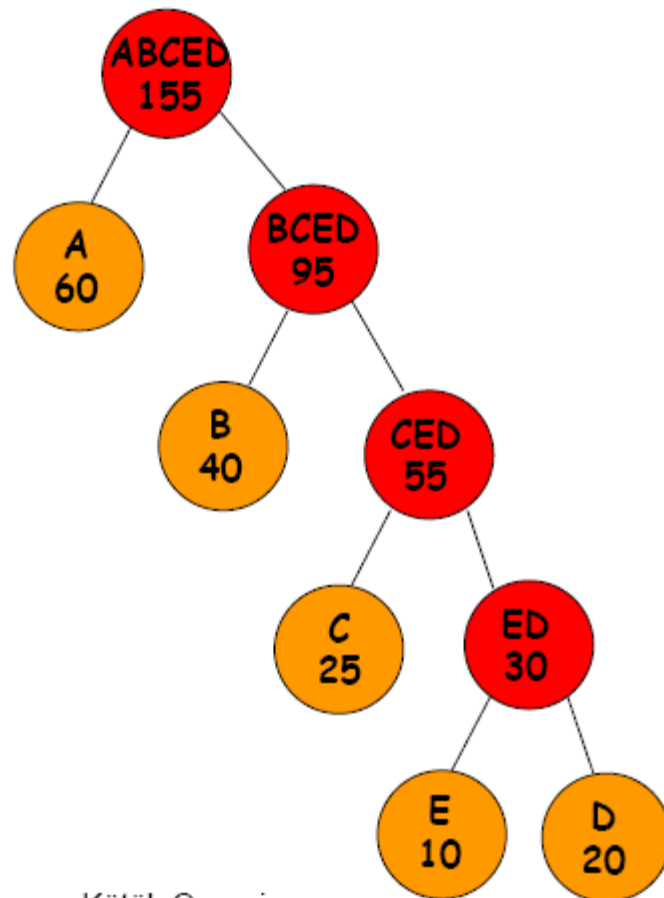
Bir önceki adımdaki işlem terkarlanılarak en küçük frekanslı 2 düğüm tekrar toplanır ve yeni bir düğüm oluşturulur.



## Adım 4



## Adım 5



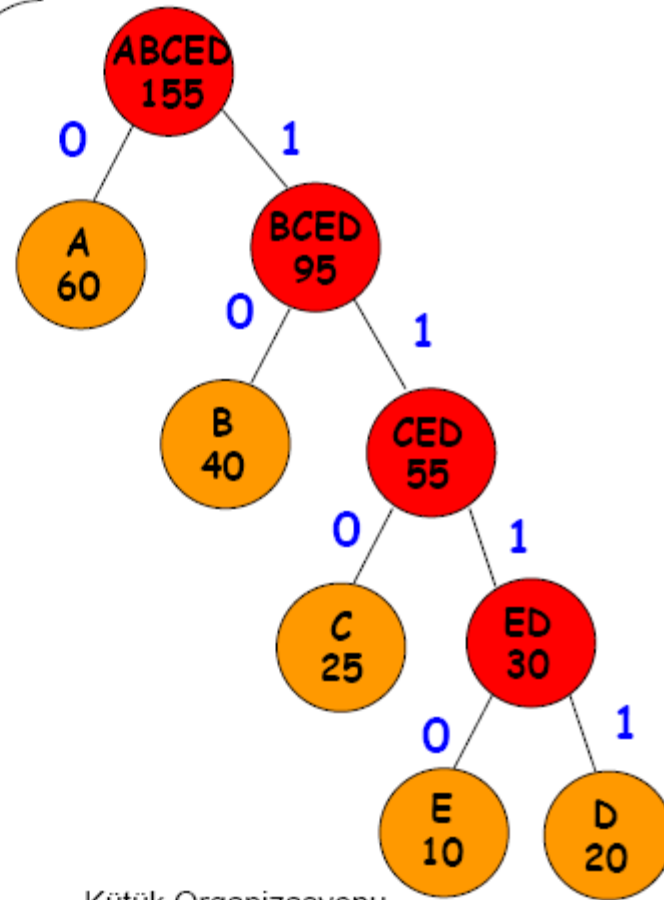
Kütük Organizasyonu



## Adım 6

Huffman ağacının kodu oluşturulurken ağacın en tepesindeki kök düğümden başlanır.

Kök düğümün soluna ve sağına giden dallara sırasıyla **0** ve **1** yazılır. Sırası seçimlidir.



## Veri Setinin Kodlanmış Hali

Sembol	Huffman Kodu	Bit Sayısı	Frekans
A	0	1	60
B	10	2	40
C	110	3	25
D	1111	4	20
E	1110	4	10

# Karşılaştırma

Veri setini ASCII kodu ile kodlanırsa, her karakter için 1 byte(8 bit)'lik alan gerektiğinden toplamda 155 byte(1240 bit)'a ihtiyaç olacaktı.

Huffman kodlamsı ile bu sayı :

$$60 \times 1 + 40 \times 2 + 25 \times 3 + 20 \times 4 + 10 \times 4 = 335 \text{ bittir.}$$

Görüldüğü gibi Huffman kodlamsı ile %27'lik bir kazanç sağlanmaktadır. Bu kazanç, veri setindeki sembollerin frekansları arttıkça daha da artmaktadır.

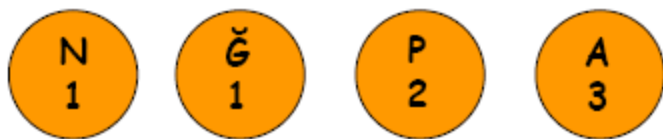
## Örnek 2

Üzerinde Huffman kodlamasını gerçekleştireceğimiz kelime “PAPAĞAN” olsun.

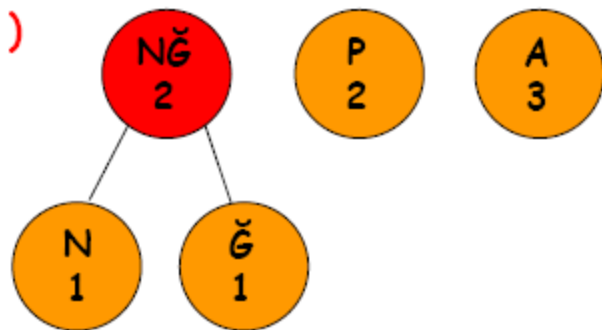
P	A	P	A	Ğ	A	N
---	---	---	---	---	---	---

Sembol	Frekans
P	2
A	3
Ğ	1
N	1

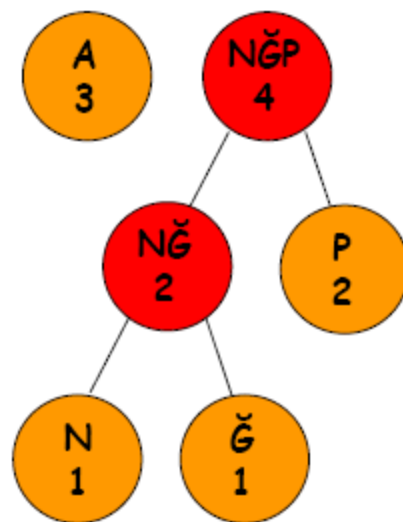
i )



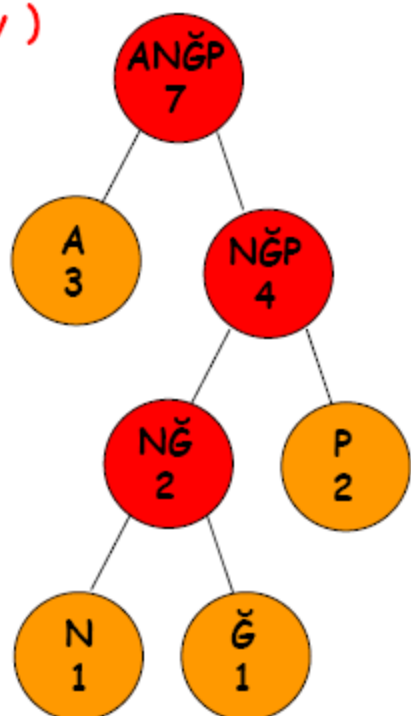
ii )



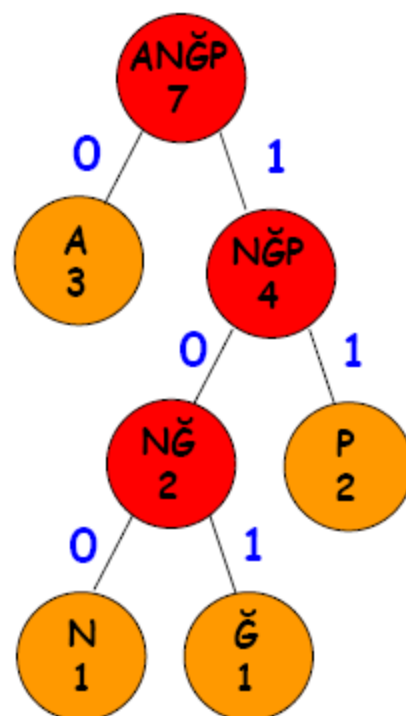
iii )



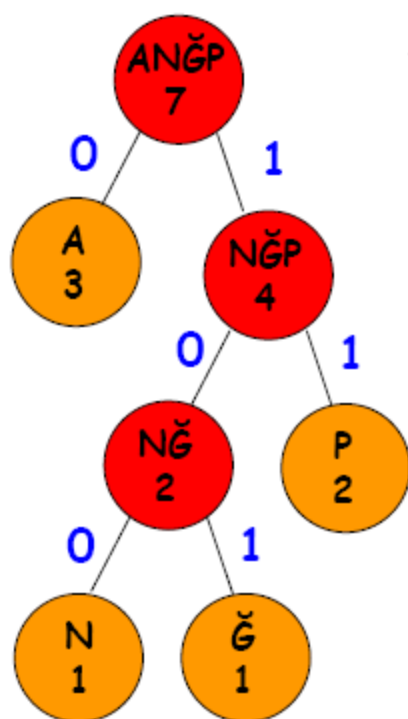
iv )



v )



Huffman Ağacı






Huffman Ağacı

Sembol	Huffman Kodu	Bit Sayısı	Frekans
P	11	2	2
A	0	1	3
Ğ	101	3	1
N	100	3	1

PAPAĞAN sözcüğünü Huffman kodu ile ifade edersek gerekli olan bit sayısı= 13 bit yeterlidir. (Ascii ile 7 byte). Huffman kodu ile kodlanmış sözcük aşağıdaki gibidir.

11 0 11 0 101 0 100  
P A P A Ğ A N

# Huffman Kodunun Çözümü

-  Frekans tablosu ve sıkıştırılmış veri mevcutsa bahsedilen işlemlerin tersini yaparak kod çözülür. Diğer bir deyişle:
-  Sıkıştırılmış verinin ilk biti alınır. Eğer alınan bit bir kod sözcüğüne karşılık geliyorsa, ilgili kod sözcüğüne denk düşen karakter yerine konulur.
-  Eğer alınan bit bir kod sözcüğü değil ise sonraki bit ile birlikte alınır ve yeni dizinin bir kod sözcüğü olup olmadığına bakılır. Bu işlem dizinin sonuna kadar yapılır. Böylece huffman kodu çözülerek karakter dizisi elde edilmiş olur.



# Run-Length Kodlama Algoritması

- Aynı byte dizilerinin sık kullanıldığı dosyalar için uygundur.
- Bir dizideki aynı değer birden fazla kez ortaya çıkarsa 3 byte ile yer değiştirir. Bu 3 byte :
  - Özel bir escape karakteri (Run-Length kodu belirtecisi. FFh)
  - Tekrar eden değer
  - Değerin tekrar etme sıklığı

# Örnek

Aşağıdaki veri seti bu algoritmaya göre kodlayalım.

22	23	24	24	24	24	24	24	24	25
26	26	26	26	26	26	25	24		

} Veri Seti

Kodlama sonucunda:

22	23	FF	24	07	25	FF	26	06	25	24
----	----	----	----	----	----	----	----	----	----	----

 elde edilir.

Veri setinin boyutu kodlama sonucunda 18 byte'tan 11 byte'a düşmüştür.

# Lempel-Ziv Kodlaması

- Lempel-Ziv kodlamasının birçok farklı türü bulunmaktadır.
- Bu kısımda LZ78'i inceleyeceğiz.
- Unix ortamındaki zip-unzip komutları ile compress-uncompress komutları bu kodlamayı kullanmaktadır.

# Örnek 1

- 2 harften meydana gelen bir alfebe olsun.

aaababbbbaaabaacaaaaabaabb



Kural

Karakter seti bizim daha önceden hiç görmediğimiz en küçük parçacıklara ayrılır.

a|aa|b|ab|bb|aaa|ba|aaaa|aab|aabb

1. a harfi
2. a harfini daha önceden gördük şimdiki harf aa
3. b harfi
4. a harfini daha önceden gördük, şimdiki harf ab
5. b harfini daha önceden gördük, şimdiki harf bb
6. aa harfini daha önceden gördük, şimdiki harf aaa
7. b harfini daha önceden gördük, şimdiki harf ba
8. aaa harfini daha önceden gördük, şimdiki harf aaaa
9. aa harfini daha önceden gördük, şimdiki harf aab
10. aab harfini daha önceden gördük, şimdiki harf aabb

- 1'den n'e kadar indexlerimiz olsun.

0	1	2	3	4	5	6	7	8	9	10
0	a	aa	b	ab	bb	aaa	ba	aaaa	aab	aabb

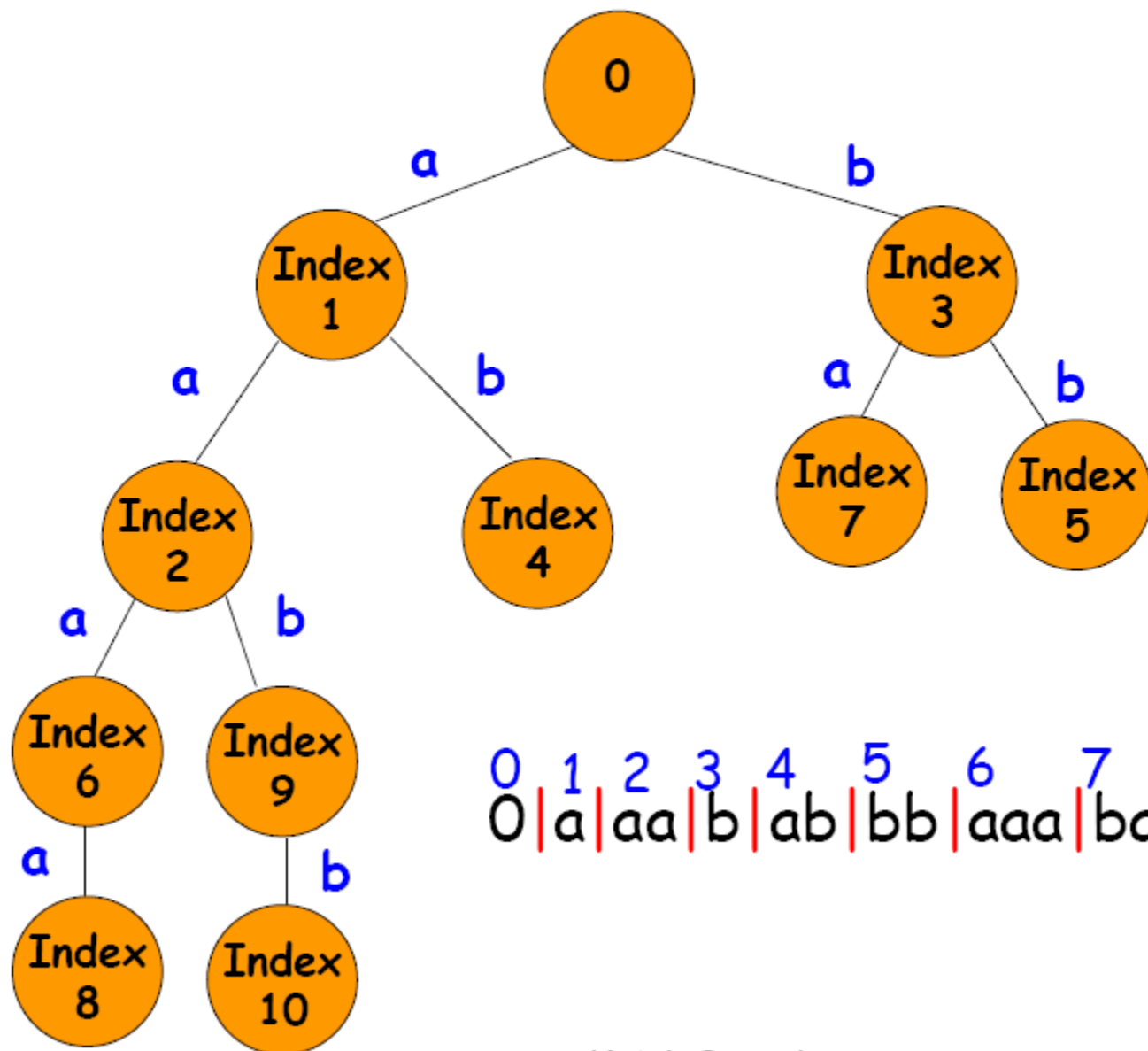
0=Null String

- Bu index yapısını kullanarak veri setini kodlayabiliriz.

1	2	3	4	5	6	7	8	9	10
0a	1a	0b	1b	3b	2a	3a	6a	2b	9b

Bu yapıda her bir parça yeni bir karakter olara algılandığından ileti, bir önceki index'in üzerine yeni bir karakter ilave edilmesi ile kodlanır.

# LZ78'in Ağaç Yapısı



## Örnek 2

Aşağıdaki karakter setini LZ78 algoritmasını kullanarak kodlayalım.

aaabbc bcdddeab



i ) Karakter setini parçalara ayıralım.

a|aa|b|bc|bcd|d|de|ab

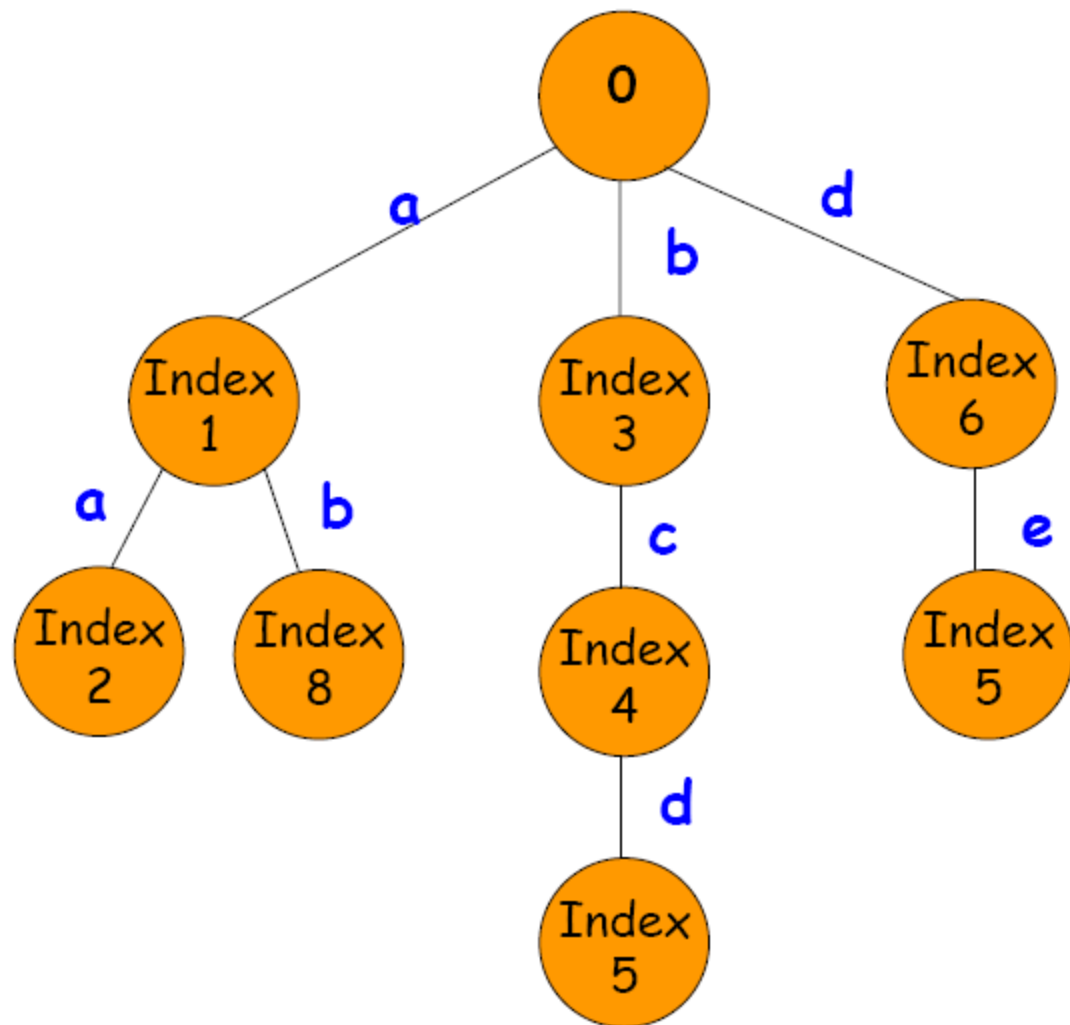
ii ) Index oluşturalım.

0 1 2 3 4 5 6 7 8  
0|a|aa|b|bc|bcd|d|de|ab

iii ) Bu indexi kullanarak veri setini kodlayalım.

|0a|1a|0b|3c|4d|0d|6e|1b

iv ) Kodlanan veri setinin ağacını oluşturalım



## Örnek 3

Üzerinde LZ78 kodlamasını gerçekleştireceğimiz kelime “PAPAĞAN” olsun.

P	A	P	A	Ğ	A	N
---	---	---	---	---	---	---

i ) Karakter setini parçalara ayıralım.

p|a|pa|ğ|an

ii ) Index oluşturalım.

0 1 2 3 4 5  
0|p|a|pa|ğ|an

iii ) Bu indexi kullanarak veri setini kodlayalım.

|0p|0a|1a|0ğ|2n

iv ) Kodlanan veri setinin ağacını oluşturalım

