YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

# SAYISAL RESİM SIKIŞTIRMA ALGORİTMALARI



Müh.Özgür ÇİLİNGİR

F.B.E Matematik Mühendisliği Anabilim Dalında
Hazırlanan

Y.Doç.Dr. Coşkun Güler

YÜKSEK LİSANS TEZİ

Prof.Dr. Mustafa Sivri

Doç.Dr. Ömer Gök

Tez Danışmanı : Yrd.Doç.Dr. Coşkun GÜLER

İSTANBUL,2003

**İÇİNDEKİLER**

Sayfa

## SİMGE LİSTESİ

| | |
|---|---|
| p(i) | Resim yoğunluklarının olasılık yoğunluğu fonksiyonu |
| L(i) | Kod kelimesi uzunluğu |
| $\overline{L}$ | Ortalama kod kelimesi uzunluğu |
| H(B) | Resmin entropisi |
| $\hat{p}(i)$ | Resim histogramı |
| $l_i$ | Resim satırı |
| $g_i$ | Gri seviyesi |
| $a_0$ | READ algoritması için geçiş elemanı |
| $a_1$ | READ algoritması için geçiş elemanı |
| $a_2$ | READ algoritması için geçiş elemanı |
| $b_1$ | READ algoritması için geçiş elemanı |
| $b_2$ | READ algoritması için geçiş elemanı |
| $V(0)$ | READ algoritması kod tablosunda bulunan notasyon |
| $V_R(i)$ | READ algoritması kod tablosunda bulunan notasyon |
| $V_L(i)$ | READ algoritması kod tablosunda bulunan notasyon |
| $f(n, m)$ | $(n, m)$ piksel pozisyonundaki sayısal yoğunluk fonksiyonu |
| $f(n+i,m+j)$ | $(i, j) \in A$ , A kümesi ile tanımlanan yakın komşuluktaki yoğunluk fonksiyonu |
| $L$ | Resim bilgisi, yakın komşuluktaki piksel yoğunluğu tahmin fonksiyonu |
| $\hat{f}(m,n)$ | $f_r(n-i,m-j)$ piksel değerleri üzerine yapılan öngörü fonksiyonu |
| $e(n,m)$ | Hata değer fonksiyonu |
| $f(m)$ | Resim satırı fonksiyonu |
| $\varepsilon(m)$ | $f(m)$'den bağımsız Gauss gürültü eklentisi |
| $R(k)$ | $f(m)$ sinyalinin oto-korelasyon fonksiyonu |
| $R(k, l)$ | $f(n, m)$ resminin iki boyutlu korelasyon fonksiyonu |
| **f** | $L = N$ x $M$ boyutunda bir resim veya resim bloğunu gösteren vector |
| **F** | Dönüşüm vektörü |
| **A** | Dönüşüm matrisi |
| $\sigma_k^2$ | $F(k)$ dönüşüm katsayısının varyansı |
| $q(n_k)$ | Nicelleştirici Bozulma Fonksiyonu |

**KISALTMA LİSTESİ**

| | |
|---|---|
| MB | Mega Byte |
| HDTV | High Defined Television |
| PCM | Pulse Code Modulation |
| PDF | Probability Density Function |
| EOL | End Of Line |
| CCITT | Consultative Comitee of the International Telephone and Telegraph |
| DPI | Dot Per Inch |
| BPS | Bit Per Second |
| MUC | Make-Up Code |
| READ | Relative Element Address Disignate |
| LZW | Lempel Ziv Welch |
| TIFF | Tagged Image File Format |
| GIF | Graphics Interchange Format |
| EOI | End Of Information |
| DPCM | Differantial Pulse Code Modulation |
| NSHP | Non Symetric Half Plane |
| WHT | Walsh-Hadamard Transformation |
| DC | Discrete Cosine |
| DCT | Discrete Cosine Transformation |
| DST | Discrete Sine Transformation |
| JPEG | Joint Photographic Experts Group |
| ISO | International Organization for Standardization |
| IEC | International Electrotechnical Commission |
| JTC1 | Joint Technical Committee 1 |
| SC2 | ISO's Technical Committee no.176, Sub-committee no.2 |
| WG10 | Working Group 10 (of Software Engineering Standards) |
| SGVIII | Star Gaze VIII |
| EOB | End Of Block |

## ŞEKİL LİSTESİ

Sayfa

## ÇİZELGE LİSTESİ

Sayfa

## ÖNSÖZ

Sayısal Resim Sıkıştırma Algoritmaları üzerine yapmış olduğum bu çalışmanın özellikle bilgisayar dünyası ile ilgili, ister okul içi olsun isterse okul dışı, her türlü projeye yardımcı olacağı düşüncesini taşımaktayım. Daha geniş bir bakış açısı ile tezde ele aldığım algoritmalar üzerine yapılmış çalışmaların, düşünsel gelişimimde günlük yaşama aktarabileceğim yaşamsal çıkarımlara neden olduğu inancındayım.

Bu tezin oluşturulmasında yardımlarını esirgemeyen değerli danışmanım Yrd.Doç.Dr. Coşkun Güler'e ve ayrıca yol gösterici yardımlarından dolayı Yrd.Doç.Dr. İbrahim Emiroğlu'na teşekkür ederim.

Ağustos 2003 Özgür Çilingir

## ÖZET

Sayısal resim kodlama ve sıkıştırma teknikleri, gösterilmek ve saklanmak istenen resimler için hafıza alanını minimum değerlerde kullanma ile ilgilenir.

Sıkıştırma etkenleri; geniş yer kaplayan resimlerin depolanması ve yayımı, saniyede iletilen bilgi oranını düşürmek, yayım zamanını azaltmak.

Kayıpsız sıkıştırma teknikleri, bilgiler sıkıştırmadan dolayı hasar görebilir olduğu için, örneğin; tıbbi teşhis resimlemede, işlenmemiş resim verisini elde etmek zor olduğunda ya da hayati değer içerdiğinde kullanılır.

Kayıplı sıkıştırma teknikleri, işlenmemiş resim verisi kolayca tekrar yayınlanabilir olduğunda ya da alıcı tarafından bilgide oluşacak kayıpların ihmal edilebildiği durumlarda, örneğin; dijital televizyonlarda, telekonferanslarda kullanılabilir.

Bu tezde kayıplı ve kayıpsız sıkıştırma tekniklerinde kullanılan, Huffman Kodlaması, Run-Length Kodlaması, LZW Sıkıştırması, Değiştirilmiş READ Kodlaması, Öngörülü Kodlama, Dönüşüm Resim Kodlaması, incelenmiştir.



**Anahtar kelimeler:** Resim Sıkıştırma, Algoritma, Huffman Kodlaması, Run-Length Kodlaması, LZW Sıkıştırması.

**ABSTRACT**

Digital image coding and compression techniques and algorithms concerned with the minimization of the memory needed to represent and store images.

Compression factors are transmission and storing of large images, reduce of baud rate, reduce of transmission time.

Lossless compression techniques are used when raw image data are difficult to obtain or contain vital information that may be destroyed by comression, e.g. in medical diagnostic imaging.

Lossy compression techniques can be used when raw image data can be easily reproduced or when the information loss can be tolerated at the receiver site, e.g. in Digital Television, Teleconferencing.

In this thesis, some of the algorithms namely Huffman Coding, Run-Length Coding, LZW Compression, Modified READ Coding, Predictive Coding, Transform Image Coding that are used in lossy and lossless compression techniques have been studied.

**Keywords:** Image Process, Algorithms, Huffman Coding, Run-Length Coding, LZW Compression.

# 1. GİRİŞ

Sayısal resim kodlaması ve sıkıştırması, sayısal bir resmi temsil edecek ve depolayacak hafıza alanını asgariye indirmekle ilgilenir. Gayet iyi bilindiği gibi ham sayısal resimler oldukça büyük yer kaplarlar. Örneğin 1024 x 1024' lük renkli bir resim 3MB' lık bir saklama alanına ihtiyaç duyar. Gerekli olan hafıza mıktarı çeşitli uygulamalarda karşımıza çıkan resim saklama ve arşivleme işlerinde sorun yaratmaktadır. Örneğin ofis uygulamalarında taranmış dokümanların saklanmasında, tıbbi görüntüleme işlemlerinde, video görüntü işlemlerinde, uzaktan takipli görüntülü sistemlerde hep aynı sorunla karşılaşılır. Resim tabanlı uygulamalarda resim saklanması ve geri alınması işlemlerini gerçekleştirecek olan hızlı sıkıştırma ve çözme algoritmaları ciddi önem taşımaktadır. Sayısal resim aktarımı, sayısal resim kodlama ve sıkıştırma tekniklerinin elzem olduğu bir başka uygulama alanıdır. Sıkıştırılmış resimler daha kısa sürede aktarılırlar. Sıkıştırma tekniklerinin kullanılması tıbbi resim aktarımı, video konferans ve fax iletimi uygulamalarında ciddi oranda tasarruf sağlar. Hızlı sıkıştırma-çözme algoritmaları ve mimarileri gerçek zamanlı sıkıştırma ihtiyacı olan video konferans ve HDTV yüksek tanımlı televizyon gibi uygulamalarda oldukça gereklidir. Çoğu iletişim kanalı gürültü barındırdığından, algoritmaların gürültü ortamında da sağlam sıkıştırma çözme yapabilmeleri de önem taşımaktadır.

Sayısal resim sıkıştırma teknikleri iki alanda incelenebilirler, kayıplı ve kayıpsız sıkıştırma algoritmaları. Kayıpsız sıkıştırma algoritmaları, ham resmin elde edilmesi güç olduğu ve sıkıştırmayla kaybedilebilecek önemli detaylar içerdiği tıbbi teşhis resimleri gibi alanlarda kullanılırlar. Kayıplı algoritmalar ise resmin kolayca tekrar oluşturulabileceği ve alıcı tarafında kayıplardan ötürü oluşabilecek bozulmaların önem arz etmeyeceği uygulamalarda kullanılırlar. Bu duruma standart bir örnek video konferans ve HDTV'de nihai alıcı insan gözüdür ve bilindiği gibi göz kimi resim bozukluklarını önemsemeyebilir. Örneğin renk bilgisinde oluşan hafif kayıplar renkli resimlerde çoğu zaman hoşgörülebilir. Oysa ki kenar bulanıklaştırma, resim sekansı sönümlendirme gibi diğer kimi bozulmalar kabul edilemez nitelikte olabilir. Bu yüzden görünen resim değişimlerini azaltmak üzere bir resim kodlayıcı tasarlanırken insanın görsel algılama kriterleri baz alınır.

Bütün sayısal resim sıkıştırma teknikleri genelde tüm sayısal resimlerde sıklıkla tekrarlanan

bilginin tespiti üzerine temellendirilmiştir. Bu tekrarlanmanın kökeni resim verisinin istatistikleridir. Resimde gereksiz bilgi tekrarı birçok yolla açıklanabilir. Bu yolların herbiri bizi başka bir sayısal resim sıkıştırma algoritması sınıfına götürür. İstatistiksel gereksiz bilgi tekrarı yaklaşımı direkt olarak resim verisi olasılık dağılımı ile ilgilidir ve resim entropisi kavramları kullanılarak bilgi teorisi teknikleriyle işlenir. Bu kavramın kullanılmaması durumunda kayıpsız sıkıştırma tekniklerine ulaşılır. Bunlara Huffman Kodlaması, Run-Length Kodlaması gibi faks iletiminde kullanılan kodlamalar örnek olarak verilebilir. Resimdeki gereksiz tekrarların tanımlanmasının bir başka yolu ise, yerel resim komşuluğunda öngörülebilirliği kullanmaktır. Yerel öngörme modelleri sağlam uzay ilişkilerinden faydalanır ve yerel komşuluktaki pikselleri öngörmeye çalışır. Öngörme modelinin iletimi veya depolanması veya ortaya çıkarılmış resim, resim sıkıştırması sonucunu doğurur. Birçok öngörülü sıkıştırma yapıları kayıplı resim sıkıştırma ile sonuçlanır. Nihayet, resim sıkıştırması, resim dönüştürmesi yöntemi ile bilgi paketleme sayesinde başarılır. Bu, soyut kosinüs dönüşümü, Karhunen-Loeve dönüşümü, soyut sinüs dönüşümü, Walsh-Hadamard dönüşümü gibi bazı ortogonal dönüşümlerin, resim enerjisini birkaç dönüşüm katsayısına yoğunlaştırılabilmesi durumunu kullanarak elde edilir. Böylece dönüşüm katsayılarının kodlaması önemli veri sıkıştırmasına yardımcı olur. Dönüşüm kodlaması algoritmaları genelde kayıplı sıkıştırma kalıplarında kullanılırlar. Dönüşüm sıkıştırması büyük oranda veri azaltmasını sağlayabildiği gibi bilinen en iyi kodlama tekniklerinden biridir.

## 2. HUFFMAN KODLAMASI

Herbir piksel için B adet bit kullanılarak dijitize edilmiş N x M boyutunda bir resim ele alalım. Resim PCM sistemine göre kodlanmış olsun. Resmi işlemeden iletmek için N x M x B adet bite ihtiyaç duyulur. Bu durumda her $2^B$' lik resim yoğunluk seviyesi B adet bit kullanarak iletilir. Piksel başına düşen ortalama bit sayısı; farklı resim yoğunluklarına farklı uzunluktaki bit gruplarının ikili olarak kodlanmasıyla azaltılabilir. p(i); $0 \le i < 2^B$ için resim yoğunluklarının pdf'i olsun. p(i) pdf'i dijital resim histogramının hesaplanmasıyla tahmin edilebilir. Bir kez pdf bilindikten sonra tekrarlanma olasılığı yüksek yoğunluklar için kısa kod kelimeleri, daha az tekrarlanan yoğunluk seviyeleri için de uzun kod kelimeleri kullanabiliriz. Bu kodlama şemasına entropi kodlaması denir. L(i) uzunluğundaki bir kod kelimesinin i, $0 \le i < 2^B$ yoğunluk seviyesi için kullanıldığını varsayalım. Kod kelimesi uzunluğu ortalaması:

$$\overline{L} = \sum_{i=0}^{2^B-1} L(i)p(i) \tag{2.1}$$

ile verilir.

L(i)' nin uzunluğu $\overline{L}$ minimum olacak şekilde seçilmelidir. $\overline{L}$ ortalama kod kelimesi uzunluğu için bir alt sınır verir:

$$\overline{L} \ge H(B) \tag{2.2}$$

Burada H(B) resmin entropisidir:

$$H(B) = -\sum_{i=0}^{2^B-1} p(i)\log_2 p(i) \tag{2.3}$$

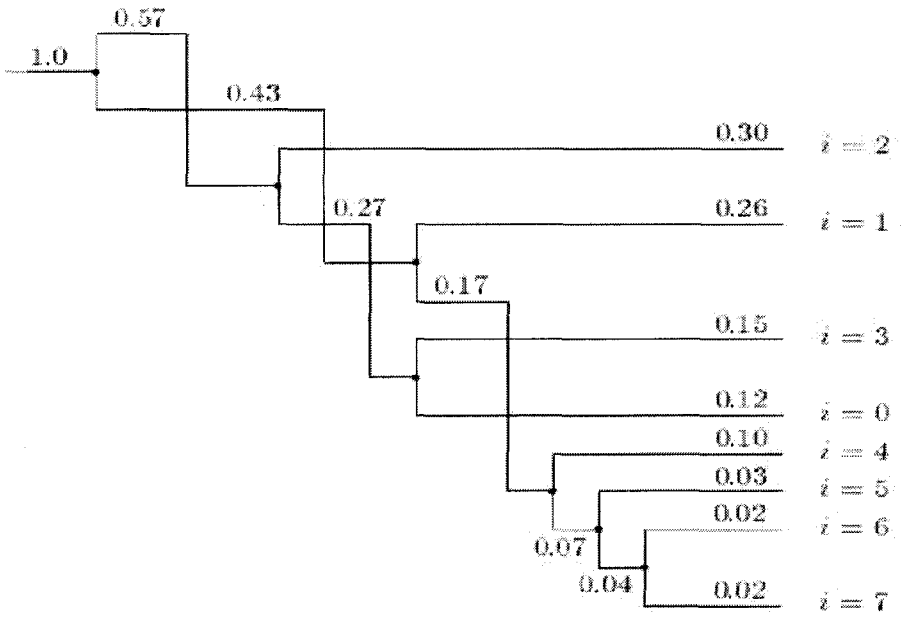
$\hat{p}(i)$ resim histogramı eğer normal dağılıma yakın bir durum arz ediyorsa resim entropisi maksimum değerine ulaşır. Bu durumda, resim daha az ağdalıdır ve ortalama kod kelimesi uzunluğu $\overline{L}$ nispeten büyüktür. Aslında p(i) bir normal dağılım ise $p(i) = 2^{-B}$, $0 \le i \le 2^B - 1$, H(B) entropisi ve ortalama kod kelimesi uzunluğu B' ye eşittir. Bu durumda entropi kodlamasından hiçbir fayda elde edilememektedir. Eğer yoğunluk seviyeleri büyük

oranda öngörülebilir ise yani kimi yoğunluk seviyeleri yüksek p(i) olasılıklarına sahiplerse, entropi kodlaması $\overline{L}$' yi azaltmakta işe yarar. Bu durum ikili modal ya da çoklu modal histogramlı resimlerde olur.

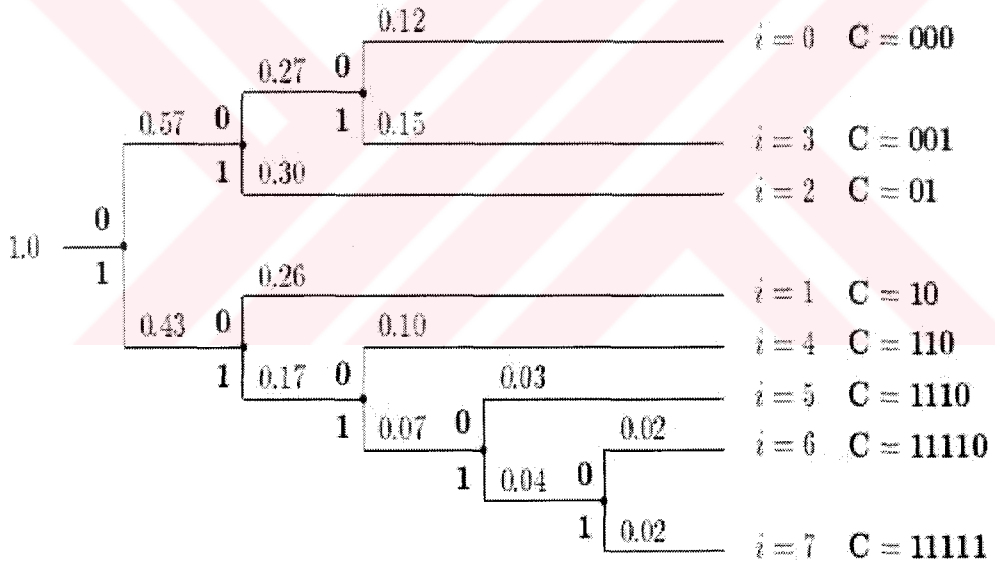
Eğer yoğunluk seviyeleri değişken kod kelimesi uzunlukları kullanarak kodlanırsa, kod kelimeleri bir ikili veri katarı oluşturmak üzere birleşirler. Bu uygulamada veri katarı alıcı tarafında çözülür. Bu yüzden birleşmiş kod kelimelerinin kombinasyonları çözülebilir olmalıdır. Huffman kodlaması bu özelliğe sahiptir. Ortalama kod kelimesi uzunluğu resim entropisine oldukça yakındır:

$$H(B) \leq \overline{L} \leq H(B)+1 \qquad (2.4)$$

Huffman kodlamasında, hiçbir kod kelimesi diğer bir kod kelimesinin öneki olamaz. Bu olgu, kodlanmış ikili veri katarının çözülebilir olmasını sağlar. Ortaya çıkan kod ağaç yapısındadır. Huffman kodlaması Şekil 2.1' de gösterildiği gibi ağaç kullanmak suretiyle oluşturulabilir. Resmin 8 yoğunluk seviyesi olduğunu varsayalım. PCM kodlaması için bit sayısı B=3' tür. p(i), i=0,..., $2^B-1$, olasılıklarının bilindiğini kabul edelim ve buna göre Şekil 2.1.a da gösterildiği gibi azalan olasılıklarla bir yoğunluk seviyeleri sütunu oluşturalım. Bu sütunun yoğunluk bilgileri Huffman kodlaması ağacının yapraklarını oluşturur. Ağaç B-1 adımda oluşur. Her adımda, en küçük olasılıklara sahip iki düğüm bir orta düğüm oluşturmak üzere bağlanırlar. Bu düğüme atanan olasılık, iki dalın olasılıklarının toplamı kadardır. Bu işlem bütün dallar (yoğunluk seviyeleri) kullanılana ve toplam olasılık 1 olana kadar sürer. Şekil 2.1.b' de görüldüğü gibi dalların üzerinden geçişleri engellemek üzere ağaç düzeltilir. Kod kelimeleri, çözümleme ağacını kökten yapraklara doğru işlerken oluşturulurlar.




(a)




(b)

Şekil 2.1 (a) Huffman kodlaması ağacının oluşturulması (b) ağacın yeniden düzenlenmesi Pitas, I., (2000)

Her seviyede üst dala 0 alt dala 1 ataması yapılır. Bu işlem ağacın tüm yaprakları elden geçirilene kadar tekrarlanır. Her yaprak bir tek yoğunluk seviyesine tekabül eder. Her

yoğunluk için kod kelimesi kökten bu yaprağa kadar olan yolun üzerindeki 0 ve 1 lerden oluşur. Ortaya çıkan çözümleme ağacı ve kod listesi Şekil 2.1.b de gösterilmiştir. Beklenildiği gibi, yoğunluk seviyeleri i=1,2' ye oluşma olasılıkları yüksek olduğundan küçük kod kelimeleri atanmıştır.

Huffman kod kelimesi büyüklüğü $L = 2^B$ ve en uzun kod kelimesi L'den fazla bite sahip olabilir. L'nin göreceli büyük olduğu durumlarda bazı kod kelimeleri oldukça uzundur. Huffman kodunda yapılabilecek pratik bir değişiklik onu $L_1 < L$ olacak şekilde kesmektir. İlk $L_1$ yoğunluk seviyesi Huffman kodlaması ile yapılır. Geri kalan yoğunluk seviyeleri, arkasından sabit uzunlukta bir kod gelen önek koduyla ifade edilirler. Bu kodlama şablonu "Kesilmiş Huffman Kodlaması" adını alır.

Huffman kodlamasında bir başka değişiklik; $0 \le i \le L-1$ olacak şekilde i yoğunluk seviyesinin aşağıdaki şekilde ifade edilmesinden de elde edilebilir:

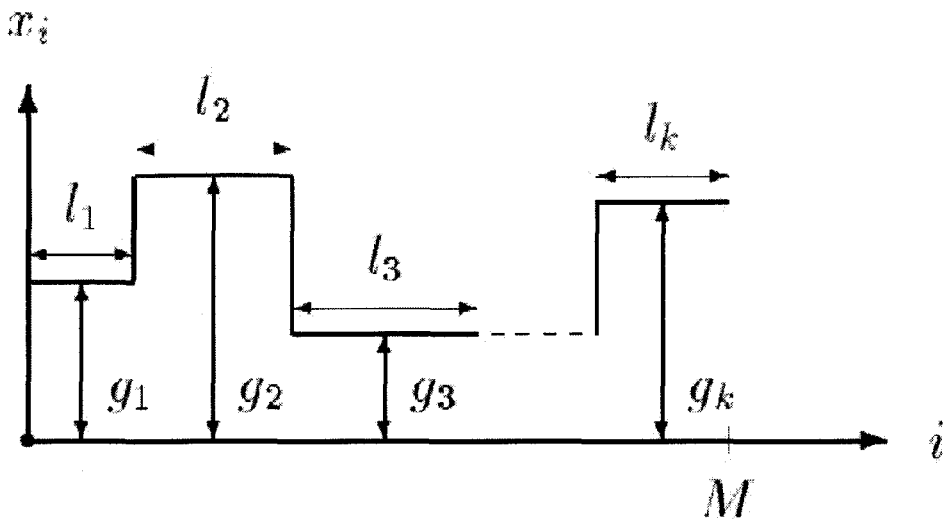
$$i = qL_1 + r,\ 0 \le q \le \left\lfloor \frac{L-1}{L_1} \right\rfloor,\ 0 \le r \le L_1 - 1 \qquad (2.5)$$

$0 \le i \le L_1 - 1$ olacak şekildeki ilk $L_1$ yoğunluk Huffman kodlaması ile işlenmiştir. Geriye kalan yoğunluklar sonlarında bir sonlandırıcı kod bulunan bir önekle, q bölümünü ifade eden, kodlanmışlardır. Sonlandırıcı kod $0 \le i \le L_1 - 1$ yoğunlukları için kullanılan Huffman kodu ile aynıdır. 2.5' teki kalan r' yi temsil eder.

Resim dizileri, yani görüntü, histogramı; nispeten geniş bir zaman aralığı için değerlendirildiğinde düzgün dağılım eğilimi gösterir. Halbuki tek çerçeveli resimlerin histogramları düzgünlükten fazlasıyla saparlar.

Bu yüzden Huffman kodlaması ham resim verisi kodlaması için en iyi çözüm değildir ve sonuçta çok iyi veri ekonomisi yapılmış olunmaz. Ama yine de diğer kodlama metodları ile birlikte kullanıldığında, yani Run-Length Kodlaması, gri tonlu resim kodlarını dönüştürür.

## 3. RUN-LENGTH KODLAMASI




Şekil 3.1 Bir resim satırının grafik gösterimi. Pitas, I., (2000)

$(x_1, x_2, ..., x_M)$ dizisinin, Şekil 3.1'de gösterildiği gibi, bir resim satırının pikselleri olduğunu kabul edelim. Her resim satırı $l_i$ uzunluğundaki ve $g_i$ gri seviyeli $1 \leq i \leq k$ olacak şekilde $k$ adet segmentten oluşur. Böylece $1 \leq i \leq k$ olacak şekilde $(g_i, l_i)$ ikilileri ile gösterilir.

$$(x_1, x_2, ..., x_M) \rightarrow (g_1, l_1), (g_2, l_2), ..., (g_k, l_k) \tag{3.1}$$

Burada

$$g_1 = x_1 \quad g_k = x_M \tag{3.2}$$

$$\sum_{i=1}^{k} l_i = M \tag{3.3}$$

'dir.

Her $(g_i, l_i)$ ikilisi renk seviyesi ve adedi verir. (3.1)'de verilmiş olan gösterim gri seviyesi adedinin yüksek olması durumunda ciddi bir sıkıştırma ile sonuçlanır. Resim ikili olduğu

taktirde verim daha da artar. Eğer bir satırın beyaz ile başladığını kabul edersek, şu şekilde ifade edebiliriz :

$$(x_1,...,x_M) \rightarrow l_1, l_2,...,l_k \tag{3.4}$$

Böylece, sadece adetler kodlanmış olur. Ortaya çıkan kodlama şeması tek boyutludur ve binary resim sıkıştırması için kullanılabilir. Bir EOL, bir resmin başlangıcını ve aynı zamanda bir satırın sonunu ifade eder. Resmin sonu genellikle ardışık EOL karakterleri ile ifade edilir(genellikle 6 adet).

| EOL | DATA | EOL | DATA | FILL | EOL | ··· | EOL | DATA | EOL | EOL | EOL | EOL | EOL | EOL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Şekil 3.2 Renk adeti kodlamasına göre, kodlanmış bir resmin formatıdır. Pitas, I., (2000)

Sonuç resminin formatı Şekil 3.2'de gösterilmiştir. Renk adedi kodlaması CCITT tarafından standartlaştırılmış ve 3. grup kodlama kalıpları içerisine dahil edilmiştir. Bu kodlama öncelikli olarak faksimile uygulamaları için telefon şebekesi üzerinden ikili resim transferinde kullanılır. Tipik resim boyutu A4 formatındadır(210 mm, 298 mm). Tipik düşey örnekleme oranı milimetrede 3.85 satır(normal çözünürlük) ve 7.7 satırdır (yüksek çözünürlük). Yatay örnekleme oranı ise 200 dpi'dır. Bu durumda bir A4 sayfası için satır başına 1728 piksel söz konusudur. Yatay örnekleme oranı kimi uygulamalar için 400 ila 1000 dpi'a çıkartılabilir. Bir satırın veri katarı çok kısa ise belli bir minimum satır uzunluğu elde etmek üzere, sıfırlardan oluşmuş özel bir doldurma katarı eklenir. 4800 bps bir iletim hızı için satır başına 96 bitlik bir minimum standart önerilir. Doldurma verisi, alıcı-verici senkronizasyonunu ilgilendirir. Alıcı tarafından gözardı edilir. Bu yüzden, renk adedi kodlaması sayısal ikili resim depolaması kullanıldığında gözardı edilir. Renk adetleri her resim için özel bir dağılım olasılığına sahiptir. Bu pdf, renk adedi kodlaması için bir huffman kodu üretmekte kullanılır.İdealde özel bir Huffman kodu her bir ikili resim için oluşturulmalıdır, ama pratikte kodlama tablosunun oluşturulması ve iletimi sırasında oluşacak zaman kayıplarından ötürü kabul edilebilir değildir. CCITT, faksimile iletiminde tipik özellikler gösteren 8 adet test dokümanı seçmiş ve bu doküman istatistiklerine göre değiştirilmiş bir Huffman kodu oluşturulmuştur. Oluşan kod tablosu Çizelge 3.1'de gösterilmiştir. Siyah ve beyaz bölümler için farklı kod kelimeleri

kullanılmıştır. İlk 64 değer Huffman kodu ile gösterilmiş ve iletim için kullanılan bir sonlandırıcı kod kelimesi içermektedir. 64 ila 1728 arasında olan uzunluklar, bir sonlandırma kodu ile biten MUC kelimesi kullanılarak iletilir. MUC, k =1, ..., 27 olacak şekilde 64k formunda bir tamsayı ile gösterilir. Burada MUC kodlanacak olan renk adedi *l*'ye eşit veya *l*'den daha azdır. A3 formuna kadar olan dokümanları kodlamak üzere geliştirilmiş bir Huffman kod tablosu oluşturulmuştur. Bu boyutta satır başına 2560 piksel vardır. Bir de kod tablosu EOL için özel bir kod kelimesini içerir.

Çizelge 3.1 Run-Length Kodlaması için değiştirilmiş Huffman kod tablosu (CCITT)

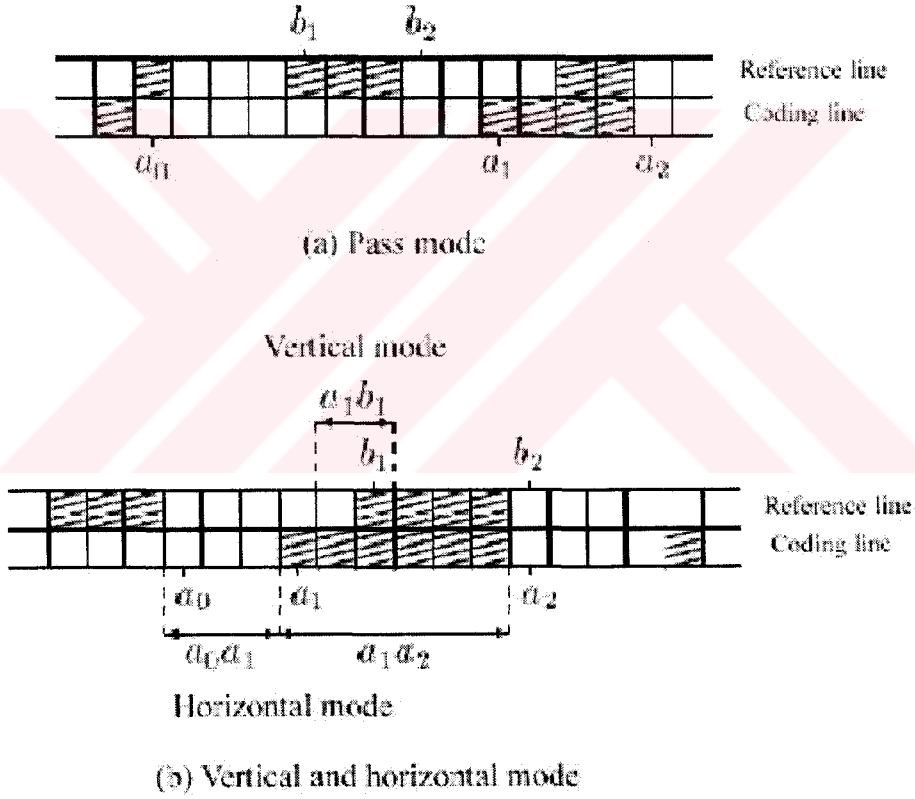
| Terminating codewords | | |
|---|---|---|
| Run length | White run | Black run |
| 0 | 00110101 | 0000110111 |
| 1 | 000111 | 010 |
| 2 | 0111 | 11 |
| 3 | 1000 | 10 |
| 4 | 1011 | 011 |
| 5 | 1100 | 0011 |
| 6 | 1110 | 0010 |
| 7 | 1111 | 00011 |
| 8 | 10011 | 000101 |
| 9 | 10100 | 000100 |
| 10 | 00111 | 0000100 |
| 11 | 01000 | 0000101 |
| 12 | 001000 | 0000111 |
| 13 | 000011 | 00000100 |
| 14 | 110100 | 00000111 |
| 15 | 110101 | 000011000 |
| 16 | 101010 | 0000010111 |
| 17 | 101011 | 0000011000 |
| 18 | 0100111 | 0000001000 |
| 19 | 0001100 | 00001100111 |
| 20 | 0001000 | 00001101000 |
| 21 | 0010111 | 00001101100 |
| 22 | 0000011 | 00000110111 |
| 23 | 0000100 | 00000101000 |
| 24 | 0101000 | 00000010111 |
| 25 | 0101011 | 00000011000 |
| 26 | 0010011 | 000011001010 |
| 27 | 0100100 | 000011001011 |
| 28 | 0011000 | 000011001100 |
| 29 | 00000010 | 000011001101 |
| 30 | 00000011 | 000001101000 |

| | | |
|---|---|---|
| 31 | 00011010 | 000001101001 |
| 32 | 00011011 | 000001101010 |
| 33 | 00010010 | 000001101011 |
| 34 | 00010011 | 000011010010 |
| 35 | 00010100 | 000011010011 |
| 36 | 00010101 | 000011010100 |
| 37 | 00010110 | 000011010101 |
| 38 | 00010111 | 000011010110 |
| 39 | 00101000 | 000011010111 |
| 40 | 00101001 | 000001101100 |
| 41 | 00101010 | 000001101101 |
| 42 | 00101011 | 000011011010 |
| 43 | 00101100 | 000011011011 |
| 44 | 00101101 | 000001010100 |
| 45 | 00000100 | 000001010101 |
| 46 | 00000101 | 000001010110 |
| 47 | 00001010 | 000001010111 |
| 48 | 00001011 | 000001100100 |
| 49 | 01010010 | 000001100101 |
| 50 | 01010011 | 000001010010 |
| 51 | 01010100 | 000001010011 |
| 52 | 01010101 | 000000100100 |
| 53 | 00100100 | 000000110111 |
| 54 | 00100101 | 000000111000 |
| 55 | 01011000 | 000000100111 |
| 56 | 01011001 | 000000101000 |
| 57 | 01011010 | 000001011000 |
| 58 | 01011011 | 000001011001 |
| 59 | 01001010 | 000000101011 |
| 60 | 01001011 | 000000101100 |
| 61 | 00110010 | 000001011010 |
| 62 | 00110011 | 000001100110 |
| 63 | 00110100 | 000001100111 |
| Make-Up Code Words | | |
| Run length | White run | Black run |
| 64 | 11011 | 0000001111 |
| 128 | 10010 | 000011001000 |
| 192 | 010111 | 000011001001 |
| 256 | 0110111 | 000001011011 |
| 320 | 00110110 | 000000110011 |
| 384 | 00110111 | 000000110100 |
| 448 | 01100100 | 000000110101 |
| 512 | 01100101 | 0000001101100 |
| 576 | 01101000 | 0000001101101 |
| 640 | 01100111 | 0000001001010 |

| | | |
|---|---|---|
| 704 | 011001100 | 0000001001011 |
| 768 | 011001101 | 0000001001100 |
| 832 | 011010010 | 0000001001101 |
| 896 | 011010011 | 0000001110010 |
| 960 | 011010100 | 0000001110011 |
| 1024 | 011010101 | 0000001110100 |
| 1088 | 011010110 | 0000001110101 |
| 1152 | 011010111 | 0000001110110 |
| 1216 | 011011000 | 0000001110111 |
| 1280 | 011011001 | 0000001010010 |
| 1344 | 011011010 | 0000001010011 |
| 1408 | 011011011 | 0000001010100 |
| 1472 | 010011000 | 0000001010101 |
| 1536 | 010011001 | 0000001011010 |
| 1600 | 010011010 | 0000001011011 |
| 1664 | 011000 | 0000001100100 |
| 1728 | 010011011 | 0000001100101 |
| 1792 | 00000001000 | 00000001000 |
| 1856 | 00000001100 | 00000001100 |
| 1920 | 00000001101 | 00000001101 |
| 1984 | 000000010010 | 000000010010 |
| 2048 | 000000010011 | 000000010011 |
| 2112 | 000000010100 | 000000010100 |
| 2176 | 000000010101 | 000000010101 |
| 2240 | 000000010110 | 000000010110 |
| 2304 | 000000010111 | 000000010111 |
| 2368 | 000000011100 | 000000011100 |
| 2432 | 000000011101 | 000000011101 |
| 2496 | 000000011110 | 000000011110 |
| 2560 | 000000011111 | 000000011111 |
| EOL | 000000000001 | 000000000001 |

# 4. DEĞİŞTİRİLMİŞ READ KODLAMASI

Run-Length Kodlaması tek boyutlu bir kalıptır ve ardışık resim satırları arasındaki düşey ilişkileri hesaba katmaz. Bu ilişki READ kodlamasında incelenmiştir. Bu iki boyutlu bir kodlama şemasıdır ve bir binary resim satırının bir önceki satırı referans alarak tanımlar. İlk resim satırı klasik Adetli Renk tek boyutlu kodlama şemasına göre kodlanabilir. Tek boyutlu kodlama bir satırda oluşan hataların resmin geri kalanına yayılmasını engellemek için her k adet ardışık satırda gerçekleştirilir. Normal çözünürlükte ve yüksek çözünürlükte taranmış dokümanlar için k değeri sırası ile 2 ve 4 olarak alınabilir. Değiştirilmiş READ kodu 3. grup faksimile iletileri için CCITT tarafından standartlaştırılmıştır.




Şekil 4.1 Değiştirilmiş READ kodlamasında geçiş elemanları. Pitas, I., (2000)

Değiştirilmiş READ kodlama kalıbı Şekil 4.1'de gösterildiği gibi, işlem yapılan satırda, $a_0$, $a_1$, $a_2$ geçiş elemanlarını ve referans satırında ise $b_1$, $b_2$ geçiş elemanlarını kullanır. Bu aşamada referans satırı hali hazırda kodlanmıştır. $a_0$'ın ilk pozisyonu işlem yapılan satırdaki ilk

pikselin soluna doğru olan beyaz geçiş pikselidir. $a_1$ ve $a_2$ aktif satırdaki $a_0$'ın sağına doğru olan geçiş çiftidir.$b_1$ ve $b_2$ referans satırındaki $a_0$'ın sağına doğru olan geçiş çiftidir. $b_1$ geçişi $a_0$ geçişine zıt renktedir. Artık kodlanacak güncel pozisyon $a_1$'dir ve kodlama $b_1$, $b_2$ ve $a_0$ referans alınarak yapılacaktır. Burada 3 muhtemel kodlama modu kullanılabilir. Kod tablosu Çizelge 4.1'de verilmiştir.

Çizelge 4.1 Değişmiş READ kodunun kod tablosu.

| Mod | Kodlanacak, değişen elemanlar | Notasyon | Kod Kelimesi |
|---|---|---|---|
| Geçiş | $b_1$, $b_2$ | $P$ | 0001 |
| Horizontal | $a_0a_1$, $a_1a_2$ | $H$ | 001 + $M(a_0a_1)$ + $M(a_1a_2)$ |
| Vertical | $b_1$'in hemen altındaki $a_1$, $a_1b_1$=0 | V(0) | 1 |
| | $a_1$, $b_1$'in sağında, $a_1b_1$=1 | $V_R(1)$ | 011 |
| | $a_1$, $b_1$'in sağında, $a_1b_1$=2 | $V_R(1)$ | 000011 |
| | $a_1$, $b_1$'in sağında, $a_1b_1$=3 | $V_R(1)$ | 0000011 |
| | $a_1$, $b_1$'in solunda, $a_1b_1$=1 | $V_L(1)$ | 010 |
| | $a_1$, $b_1$'in solunda, $a_1b_1$=2 | $V_L(1)$ | 000010 |
| | $a_1$, $b_1$'in solunda, $a_1b_1$=3 | $V_L(1)$ | 0000010 |
| | EOL | | 000000000001 |
| | bir sonraki satırın kodu 1-d | | EOL + '1' |
| | bir sonraki satırın kodu 2-d | | EOL + '0' |

- **Geçiş Modu :** $b_2$'nin Şekil 4.1.a'da gösterildiği gibi kodlanacak olan $a_1$ pikselinin solunda olduğunu varsayalım. Bu, lokal siyah ve beyazın referanstan uzaklaştığı ve aktif satırın uymadığı anlamına gelir. Geçiş modu sadece bir kod kelimesi kullanır, Tablo 4.1'de görülebileceği gibi(0001). Bir sonraki kodlamaya geçmeden evvel, $a_0$ pikseli $b_2$ ve $b_1$'in altında eşitlenir, $b_2$ güncellenir.

- **Düşey Mod :** $b_2$'nin $a_1$'in sağında olduğunu varsayalım. Ayrıca $b_1$'in de $a_1$'den sağa veya sola doğru 3 piksellik komşuluk içinde olduğunu varsayalım. Böylece $a_1$, $b_1$'den referans alınarak kodlanabilir. $a_1$, $b_1$ mesafesi çeşitli muhtemel değerler alabilir. 0, sağda 1,2,3, solda 1,2,3. Düşey modda ilgili kod kelimesini Çizelge 4.1'den seçerek uygularız ($V(0)$ , $V_R(i)$, i = 1,2,3, $V_L(i)$ , i = 1,2,3 durumları). Referans pikseli $a_0$'ı aktif $a_1$'e eşitleyerek güncelleriz($a_0 = a_1$). $a_1$, $b_1$, $b_2$'yi günceller ve sonraki kodlama adımına geçeriz.

- **Yatay Mod :** Geçiş modu ve düşey mod uygulaması reddedildiğinde, yani $|a_1b_1|$ mesafesi 3 pikselden büyük olacak şekilde uzaklarsa bu durum söz konusudur. Bu durumda $a_2$'nin pozisyonu tespit edilir ve $a_0a_1$ ve $a_1a_2$ uzunlukları Çizelge 3.1'de gösterilmiş Adetli Renk kodlamada kullanılan değiştirilmiş Huffman kodu kod tablosu kullanılması ile kodlanır. Burada kod kelimesi 001 + $M(a_0a_1)$ + $M(a_1a_2)$'dir ve "+" birleştirilmeyi, "$M(.)$" Huffman tablosunu ifade eder. Yatay mod süresince $a_1$ ve $a_2$ pozisyonları kodlanır, bu yüzden yeni referans pozisyonları $a_0 = a_2$'dir. $a_1,b_1,b_2$ pozisyonları yeni bir kodlama adımına geçmeden önce güncellenirler.

Bu her 3 modda da bir satır sonuna erişildiğinde durur. A4 sayfalar için $a_0 = 1729$'dur. Eğer $k = 2,4$ adet satır 2 boyutlu yöntemle kodlanırsa sonraki satır tek boyutlu Adet Renk kodlama ile kodlanır. Adet Renk kodlama, değiştirilmiş READ kodlaması, her ikisi de öncelikle her ne kadar gri-seviyeli resim kodlaması için kimi değişiklikler ve uzantılarla kullanılacak olsalar da ikili resim sıkıştırma kalıplarıdırlar.

# 5. LZW SIKIŞTIRMASI

LZW algoritması Lempel-Ziv ve Welch tarafından genel maksatlı bir sıkıştırma kalıbı olarak ortaya atılmıştır. Her türlü ikili veri dosyasının sıkıştırılması için kullanılabilir. Şu anda piyasadaki en popüler sayısal resim sıkıştırma algoritmalarından biridir ve TIFF ve GIF gibi standart resim dosyaları ile birlikte kullanılır. Bu popülerliğini kayıpsız, hızlı, verimli olmasına ve her bit derinliğindeki resimler üzerinde işlem yapabilmesine borçludur.

LZW sıkıştırması sık rastlanan bit katarlarını çıktı kod kelimelerine bağlayan kod tablosu oluşturma esası üzerine temellendirilmiştir. Sayısal resim tek boyutlu bit katarı olarak değerlendirilir. Kodlanan resim de aynı şekilde bir bit katarı olarak ortaya çıkar. Algoritma sonuç katarını üretirken aynı zamanda kod tablosunu da günceller. Böylece kod tablosu sıkıştırılacak resmin özel karakteristiklerine göre adapte edilebilir. Kod tablosu 4096 adede kadar kod kelimesi içerebilir ve böylece her kod kelimesi 12 bite kadar uzun olabilir. İlk 256 kod kelimesi [0,..., 255] 0 ile 255 arası sayıları içerirler. #256 kod kelimesi özel bir sıfırlama kodu içerir. #257 kod kelimesi EOI içerir. Dosyanın sonunu veya resim bölüm sonunu (resim eğer bölümlere ayrılmışsa) ifade eder. #258'den #4095'e kadar olan kod kelimeleri resim içerisinde sıkça rastlanan bit katarlarını içerir. LZW kod tablosunun yapısı Çizelge 5.1'de gösterilmiştir. İlk 512 tablo elemanının her biri 9 bit ile temsil edilir. #512'dan #1023'e kadar olan kod kelimeleri 10 bit uzunluğundadırlar. Benzer şekilde #1024'te 11 bitlik kod kelimelerine ve #2048'de de 12 bitlik kod kelimelerine geçeriz. Kod tablosunun #258'den #4095'e kadar olan ki kısmı işlem sırasında yapılandırılır.

Çizelge 5.1 LZW Kod Tablosu.

| | |
|---|---|
| 0 | 0 |
| ⋮ | ⋮ |
| 255 | 255 |
| Temizleme kodu | 256 |
| EOI kodu | 257 |
| Kelime | 258 |
| Kelime | 259 |
| ⋮ | ⋮ |
| Kelime | 4095 |

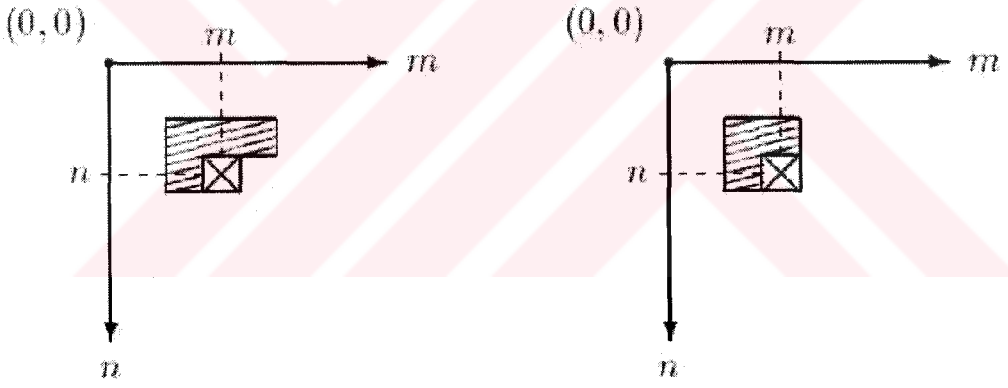
LZW sıkıştırma ve çözüm algoritmalarının hızları kod tablosunun içindeki verimli arama prosedürünün uygulamasına sıkı sıkıya bağlıdır. Bu arama rutini için çeşitli uygulamalar yapılabilir özellikle hashing veya ağaç bazlı teknikler kullanılabilir. Çözüm genelde daha hızlıdır çünkü burada arama işlemine gerek duyulmamaktadır. Çıktı string'i direkt olarak ilgili kod kelimeleri kullanılarak oluşturulur. Sıkıştırımsal bir bakış açısında sıkıştırılma oranı bire bir buçuk ila bire üç oranında başarılabilir. Genel bir deyişle sağlam bir sıkıştırma bu şekilde bitmap veya binary resimler için elde edilir. Ama çoğu durumda gri-skalalı resimlerin LZW kodlaması sağlam bir sıkıştırma işlemi olarak kabul edilmez.

# 6. ÖNGÖRÜLÜ KODLAMA

Sayısal resimlerde sıkça tekrarlanan bilgiyi tanımlamak için yakın resim komşuluğunda öngörülebilirlik kullanılır. $f(n, m)$ fonksiyonu $(n, m)$ piksel pozisyonundaki sayısal yoğunluk ve $f(n+i, m+j)$, $(i, j) \in A$, A kümesi ile tanımlanan yakın komşuluktaki yoğunluklar olsun. $f(n, m)$ piksel yoğunluğu genelde olduğu gibi resim bilgisi yakın komşuluktaki bilgi ile ilişkili ise etraftaki piksellerin yoğunluklarından yola çıkılarak tahmin edilebilir. Tahmin formülü $L$ şöyledir :

$$\hat{f}(m,n) = L(f(n-i,m-j),(i,j) \in A,(i,j) \neq (0,0)) \tag{6.1}$$

Bu fonksiyon genellikle A penceresi içindeki piksel yoğunluklarının lineer bir fonksiyonudur. A penceresinin Şekil 6.1'de gösterilen gibi bir öngörücüye sahip olduğunu varsayalım.




Şekil 6.1 Resim öngörmesinde kullanılan pencereler. Pitas, I., (2000)

Eğer bir pencere resim düzlemini satır bazında soldan sağa ve yukarıdan aşağıya tarayacak olursa, sadece geçmişe ait pikselleri yani hali hazırda taranmış pikselleri göz önünde bulundurmuş olur. Bu durumda geçmişte oluşturulmuş $f_r(n-i,m-j)$ piksel değerleri üzerine $\hat{f}(m,n)$ öngörüsü yapılabilir :

$$\hat{f}(m,n) = L(f_r(n-i,m-j),(i,j) \in A) \tag{6.2}$$

(6.2) üzerine bina edilmiş rekürsif bir öngörülü uygulama tekniği ortaya konulabilir. Optimal

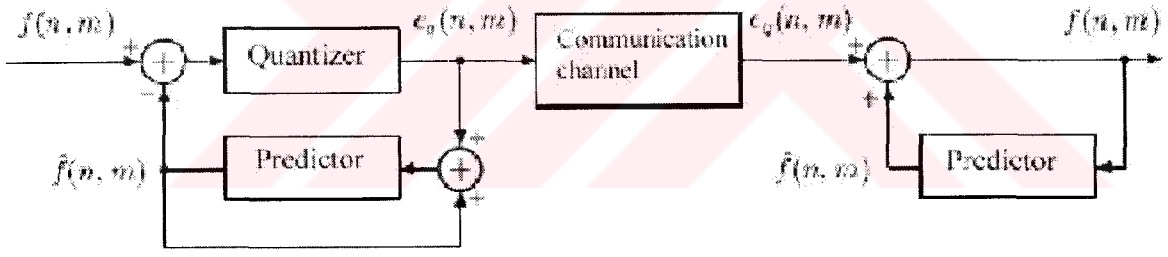
bir öngörü kuralı $L$'nin elde edildiğini ve sınır değerlerinin bulunduğunu böylece (6.2) rekürsiyonunun uygulanabileceğini varsayalım. Her adımda $f(n, m)$ piksel yoğunluğu yerine hatayı kodlamak yeterlidir :

$$e(n,m) = f(n,m) - \hat{f}(n,m) \tag{6.3}$$

$e_q(n,m)$, $e(n,m)$ hatasının kodlanmış ve sayılmış değeri ise $f(n,m)$ piksel değeri aşağıdaki gibi oluşturulur :

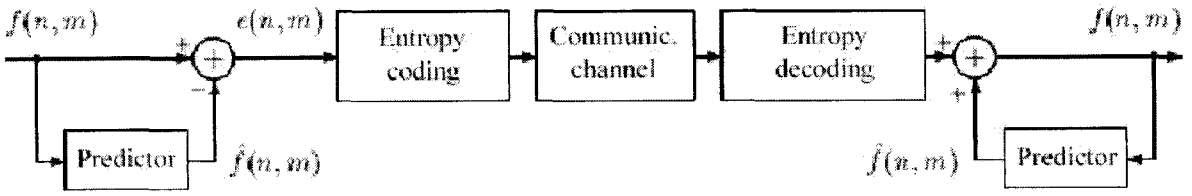
$$f_r(n,m) = L(f_r(n-i,m-j),(i,j) \in A) + e_q(n,m) \tag{6.4}$$

Eğer $\hat{f}(m,n)$ başarılı ise hata terimi $e(n,m)$ küçük bir dinamik sahası vardır ve nispeten küçük kod kelimesi kodlanarak kodlanabilir. Böylece gerçek bir sıkıştırma başarılabilir. Öngörme katsayılarının ve kodlanmış hatanın iletimi (6.4) kullanılarak $f(n,m)$'nin yeniden oluşturulması için gereklidir. Şekil 6.2'de görülebilir.




Şekil 6.2 Tahmin edilebilir DPCM. Pitas, I., (2000)

Bu kodlama şeması telekominikasyonda konuşma ve resim kodlaması için hayli kullanılır. DPCM kodlama prosesinde nicelleyici kullandığı için kayıplı bir kodlama şemasıdır. Hata sinyalinin nicelleştirilmesi her zaman geri dönülmez bir bozulma yaratır. Ama Şekil 6.3'te görüldüğü gibi bozulma yapmayan DPCM şemaları da vardır.




Şekil 6.3 DPCM entropi kodlaması ve DPCM. Pitas, I., (2000)

Öngörücü, tam sayı çıktı üretecek şekilde zorlanabilir. Bu durumda hata sinyali $e$ $(n,m)$ entropi kodlama teknikleri kullanılarak herhangi bir nicelleştirme yapılmadan kodlanır(Huffman Kodlaması). DPCM'den sonra uygulanan entropi kodlaması ile bozulmasız resim iletimi gerçekleştirilir.

DPCM'nin performansı öngörücüye ve katsayıların seçimine bağlıdır. Burada en basit yaklaşım her resim satırının tek boyutlu DPCM'sinin uyarlanmasıdır. $f(n,m)$ resim satırının (gösterim basitliği için $f(m)$ ile ifade edilen) aşağıdaki gibi modellendiğini varsayalım :

$$f(m)=\sum_{k=1}^{p}a(k)f(m-k)+\varepsilon(m)\ ,\ E\left[\varepsilon^2(m)\right]=\sigma^2 \tag{6.5}$$

Burada $\varepsilon(m), f(m)$'den bağımsız Gauss gürültü eklentisidir. (6.2) öngörme şeması için doğal bir seçim aşağıdaki gibi olacaktır :

$$\hat{f}(m)=\sum_{k=1}^{p}a(k)f_r(m-k) \tag{6.6}$$

Ögörme katsayıları $\mathbf{a}=[a(1),...,a(p)]^T$ vektörünü oluşturur. A öngörme penceresi $\{f_r(m-1),...,f_r(m-p)\}$oluşturulmuş piksellerini içerir. Nicelleştirilmiş hata sinyali

$$e_q(m)=Q[e(m)]=Q\left[f(m)-\hat{f}(m)\right] \tag{6.7}$$

alıcıya iletilir. Resim satırı (6.4) kullanılarak yeniden oluşturulur.

$$f_r(m)=\sum_{k=1}^{p}a(k)f_r(m-k)+e_q(m) \tag{6.8}$$

Öngörme katsayıları normal denklemler kümesini seçerek çözülürler.

$$\begin{bmatrix} R(0) & R(1) & \cdots & R(p-1) \\ R(1) & R(0) & \cdots & R(p-2) \\ \vdots & \vdots & \vdots & \vdots \\ R(p-1) & R(p-2) & \cdots & R(0) \end{bmatrix} \begin{bmatrix} a(1) \\ a(2) \\ \vdots \\ a(p) \end{bmatrix} = \begin{bmatrix} R(1) \\ R(2) \\ \vdots \\ R(p) \end{bmatrix} \tag{6.9}$$

Burada $R(k)$, $f(m)$ sinyalinin oto-korelasyon fonksiyonudur. Tek boyutlu öngörme modelleri iki boyutlu modellere genişletilebilir :

$$\hat{f}(n,m) = \sum_{(i,j)\in A} \sum a(i,j) f(n-i, m-j) \tag{6.10}$$

İki boyutlu öngörme modeli katsayıları ortalama kare hatasını minimize edilerek seçilirler :

$$E\left[ \left| f(n,m) - \sum_{(i,j)\in A} \sum a(i,j) f(n-i, m-j) \right|^2 \right] \tag{6.11}$$

Minimizasyon

$$R(k,l) = \sum_{(i,j)\in A} \sum a(i,j) R(k-i, l-j) \tag{6.12}$$

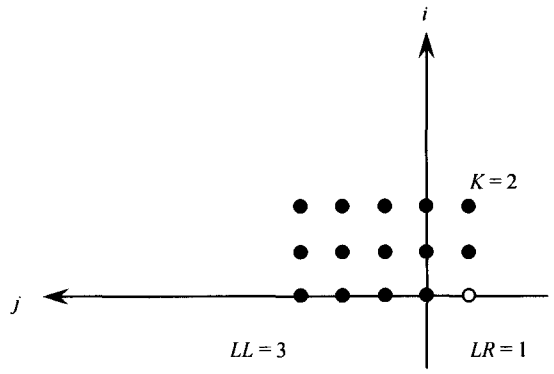
formundaki bir normal denklemler kümesinin sonucuna götürür. Burada R ( $k$, $l$ ) , $f(n , m)$ resminin iki boyutlu korelasyon fonksiyonudur. (6.12) sisteminin çözümü $a(i,j)$, $(i,j)\in A$ olacak şekilde optimal model katsayılarına götürür. Bu katsayılar bir kere bilindikten sonra hata resmi :

$$e(n,m) = f(n,m) - \hat{f}(n,m) \tag{6.13}$$

elde edilir ve nicelleştirilir. Sayısal resim (6.4) kullanılarak alıcı tarafından yeniden oluşturulur :

$$f_r(n,m) = \sum_{(i,j)\in A} \sum a(i,j) f_r(n-i, m-j) + e_q(n,m) \tag{6.14}$$

Bu bölümde Şekil 6.4'te gösterilmiş formdaki iki boyutlu, NSHP öngörme penceresi için gerekli katsayıları hesaplayan bir algoritma tanımlanacak.




Şekil 6.4 NSHP öngörme penceresi. Pitas, I., (2000)

Bu pencerenin boyutu ($LL + LR + 1$) x $K + LL + 1$ pikseldir. Şekil 6.4'te gösterilen pencere $LL = 3$, $LR = 1$ , $K = 2$ değerlerine sahiptir ve boyutu 14 pikseldir. $a$ ( $i$ , $j$ ) öngörme katsayıları ($LL + LR + 1$) x $K + LL + 1$ boyutunda tek boyutlu bir A dizisi içine konmuş olsun. Yukarıdan aşağıya doğru satır bazında yerleştirilsin : A = [$a(0,0)$,..., $a(0,LL)$, $a(1,-LR)$,..., $a(1,LL)$,..., $a(K,-LR)$,..., $a(K,LL)$]$^T$ .

Çoğu durumda küçük öngörme pencerelerini kullanmak yeterli olur.

$$\hat{f}(n,m) = a_1 f(n-1,m) + a_2 f(n,m-1) + a_3 f(n-1,m-1) + a_4 f(n-1,m+1) \tag{6.15}$$

model katsayıları $a_i$ , i = 1,...,4 ve gürültü varyansı $\sigma^2 = E[\varepsilon^2(n,m)]$ , aşağıdaki sistem çözülerek elde edilir.

$$\begin{bmatrix} R(0,0) & R(1,-1) & R(0,1) & R(0,1) \\ R(1,-1) & R(0,0) & R(1,0) & R(1,-2) \\ R(0,1) & R(1,0) & R(0,0) & R(0,2) \\ R(0,1) & R(1,-2) & R(0,2) & R(0,0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} r(1,0) \\ r(0,1) \\ r(1,1) \\ r(1,-1) \end{bmatrix} \tag{6.16}$$

$$\sigma^2 = R(0,0) - a_1 R(1,0) - a_2 R(0,1) - a_3 R(1,1) - a_4 R(1,-1) \tag{6.17}$$

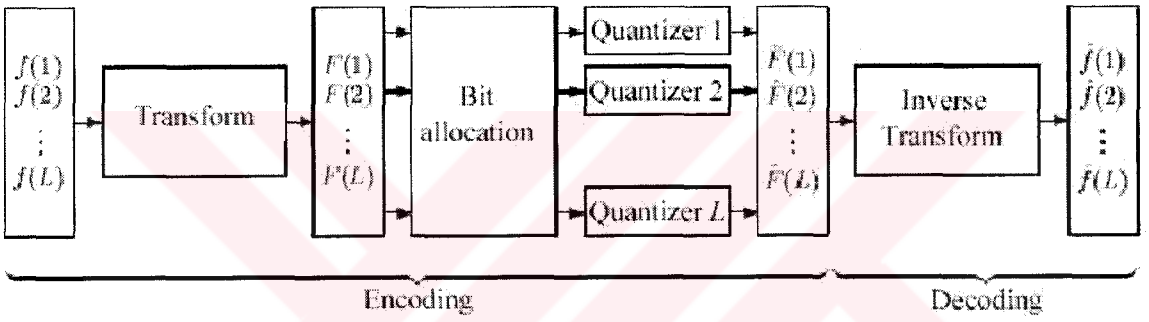
Oto-korelasyon katsayıları $R$ ( $i$ , $j$ ) (6.18) kullanılarak tahmin edilir :

$$R(i,j) = \frac{1}{(2N+1)(2M+1)} \sum_{i=-N}^{N} \sum_{j=-M}^{M} f(k,l) f(k+i, l+j) \tag{6.18}$$

Öngörülü DPCM basit bir sayısal resim sıkıştırma tekniğidir ve hem yazılım hem donanım seviyesinde implemente edilebilir. Ortalama sıkıştırma oranları elde etmek için kullanılır(sonraki bölümde anlatılacak olan dönüşüm tekniklerine oranla). Bir diğer dezavantajı ise kanal gürültüsüne karşı hassasiyetidir. Ani gürültü etkileri tüm bir satır veya hatta tüm bir resim içerisinde etkisini gösterebilir (eğer iki boyutlu öngörülü DPCM kullanılmışsa). Nihayet optimal katsayılar tüm bir resim düzlemi üzerinden elde edilmiş resim istatistikleri kullanılarak belirlenebilir. Yine de resim istatistikleri resimden resime hatta resmin farklı bölgelerinde de değişiklikler gösterir.

# 7. DÖNÜŞÜM RESİM KODLAMASI

Sayısal resim kodlamaya bir başka yaklaşım da resim enerjisini bir kaç dönüşüm katsayısında yoğunlaştıracak şekilde resim dönüşüm kodlamasını kullanmaktır. Enerji paketi elde edilirse büyük sayıdaki bir dönüşüm katsayıları grubu gözardı edilip geri kalanlar değişken uzunluktaki kod kelimeleri ile kodlanabilirler. Böylece büyük oranda veri sıkıştırması gerçekleştirilmiş olur. Ters dönüşüm kullanılarak resim çözümlemesi kolaylıkla kullanılabilir. Sayısal resimlerin dönüşüm kodlaması Şekil 7.1'de gösterilmiştir.




Şekil 7.1 Dönüşüm kodlama ve çözümü. Pitas, I., (2000)

**f** , $L = N$ x $M$ boyutunda bir resim veya resim bloğunu gösteren vektör olsun. Dönüşüm vektörü olan **F**

$$\mathbf{F} = \mathbf{Af} \tag{7.1}$$

ile verilir. Burada **A** dönüşüm matrisi olsun. Ters dönüşüm ise şu şekilde tanımlanır

$$\mathbf{f} = \mathbf{A}^{-1}\mathbf{F} \tag{7.2}$$

Bir dönüşüm eğer aşağıdaki şartı gerçekliyorsa uniterdir.

$$\mathbf{A}\mathbf{A}^{*\mathrm{T}} = \mathbf{A}^{\mathrm{T}}\mathbf{A}^{*} = \mathbf{I} \tag{7.3}$$

Uniter bir dönüşümde enerji korunur

$$\|\mathbf{f}\|^2 = \sum_{k=1}^{L} |f(k)|^2 = \sum_{k=1}^{L} |F(k)|^2 = \|\mathbf{F}\|^2 \tag{7.4}$$

Durumların çoğunda sinyal enerjisi dönüşüm katsayıları arasında eşit olmayacak şekilde dağıtılır. DC katsayısı ve diğer bazı düşük frekanslı katsayılar $F(k)$, $1 \le k \le K << L$ sinyal enerjisinin çoğunu yoğunlaştırırlar. Böylece birçok dönüşüm katsayıları (örneğin $F(k)$, $k>K$ ) çok fazla bilgi kaybı olmaksızın gözardı edilebilirler. Değişen sayıda bit $n_k$, $1 \le k \le K$ , piksel başına bitlerin ortalama sayısı önceden belirlenmiş $B$ sayısına eşit olacak şekilde geri kalan katsayılara ayrılır. Bir kere bit sayısı ayrıldıktan sonra dönüşüm katsayıları $F(k)$ $1 \le k \le K$ nicelleştirilmelidir. $K$ adet farklı nicelleştirici tasarlanmalıdır. Nicelleştiriciler tasarlandıktan sonra dönüşüm katsayılarına uygulanırlar ve sonuç kodlanmış resim veya resim bloğu olarak ortaya çıkar. Çözme işlemi oldukça basittir. Ters dönüşüm $\mathbf{A}^{-1}$ matrisi orijinal resim vektörü $\mathbf{f}$ 'in bir öngörüsü olan $\hat{\mathbf{f}}$ 'i üretmek üzere kodlanmış katsayı vektörü olan $\hat{\mathbf{F}}$ 'e uygulanır.

İyi bir dönüşüm kodlama algoritması elde edebilmek için bir çok problemi çözmek gerekir.İlk problem kullanılacak dönüşümün seçilmesidir. Bir çok alternatif mevcuttur : Ayrık Fourier dönüşümü, WHT, DCT, DST, vs... .Yoğun deneylerle gösterilmiştir ki, DCT en iyi sıkıştırma performansını verir; bu ayrıca ortaya yeni çıkan CCITT'nin JPEG sıkıştırma standardında da kullanılır. Diğer bir problem resim blok boyutunun seçimidir. Bütün bir resim için bir dönüşümün uygulanması farklı bölgelerde resim istatistiğinin değişmesinden dolayı önerilmez. Resim üstüste çakışmayan bloklara ayrılır ve herbiri bağımsız şekilde kodlanır. Tipik blok boyutu 8 x 8 veya 16 x 16'dır.

Diğer bir problem, bit tahsisinin belirlenmesidir. $n_k$, $k$ =1,...,$L$'nin herbir $F(k)$ $k$=1,..., $L$ dönüşüm katsayısı için ayrılan bit adedi olduğunu varsayalım.$n_k$ = 0 ise ilgili katsayı $F(k)$'nın gözardı edildiği anlamına gelir. Piksel başına ortalama bit adedi $B$ ise aşağıdaki ilişki sağlanır:

$$\frac{1}{L}\sum_{k=1}^{L} n_k = B \tag{7.5}$$

katsayı nicelleştirilmesine bağlı olarak ortaya çıkan ortalama bozulma aşağıdaki şekilde

verilir :

$$E = \frac{1}{L}\sum_{k=1}^{L} E\left[\left[F(k) - Q[F(k)]\right]^2\right] = \frac{1}{N}\sum_{k=1}^{L} \sigma_k^2 q(n_k) \tag{7.6}$$

Burada $Q$ [.] nicelleştirmeyi ifade eder ve $\sigma_k^2$, $F(k)$ dönüşüm katsayısının varyansıdır. Nicelleştirici *Bozulma Fonksiyonu* $q(n_k)$, $q(0)=1$ ve $q(\infty)=0$ olacak şekilde monoton azalan bir fonksiyondur. Bit tahsis işleminin amacı ise ortalama kare hatasını (7.6), (7.5) kısıtı altında minimize etmektir. Mantıklı bir bit adet seçimi aşağıdaki gibi verilebilir :

$$n_k = B + \frac{1}{2}\left[\log_2 \sigma_k^2 - \log_2\left(\prod_{k=1}^{L} \sigma_k^2\right)\right] \tag{7.7}$$

(7.7)'de yuvarlama, tamsayı bit uzunluklarını kullanmak için uygulanır. Tamsayı bit tahsisi için başka algoritmalar da vardır. Katsayı varyansları $\sigma_k^2$, $k = 1, ..., L$ çeşitli resim bloklarının dönüşümlerinden elde edilir. 16 x 16 blokluk bit tahsisine ilişkin bir örneğin kosinüs dönüşümü Şekil 7.2'de gösterilmiştir. Ortalama bit sayısı piksel başına 0.5 bittir.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 7 | 6 | 5 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 7 | 6 | 5 | 4 | 3 | 3 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 6 | 5 | 4 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 5 | 4 | 3 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 3 | 3 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Şekil 7.2 16 x 16'lık blok kosinüs dönüşümünün ikili tahsisi. Pitas, I., (2000)

Bit tahsisi elde edildiğinde her dönüşüm katsayısı için birer optimal nicelleştirici tasarımı gerçekleştirilmelidir. İdealde dönüşüm katsayılarının pdf'i biliniyor olmalıdır. DC dönüşüm katsayılarının pdf'ini modellemede Rayleigh dağılımı kullanılır. Gauss veya Laplace dağılımları ise geri kalan dönüşüm katsayılarını modellemede kullanılır. Pratikte DC katsayıları için düzgün nicelleştirme kullanılır. Bu katsayılar için maksimum bit uzunluğu kullanılır (genellikle $n_1 = 8$ bit). Bütün diğer $F(k)$ katsayıları birim varyansa sahip katsayıları elde etmek üzere ilgili $\sigma_k^2$ varyanslarına bölünürler. Birim Gauss dağılımları için optimal nicelleştiriciler daha sonra oluşturulur. Her $n$, $1 \le n \le n_1$ için bir nicelleştirici oluşturulur. Eğer maksimum bit uzunluğu 8 ise 7 nicelleştiriciye ihtiyaç vardır. Bu nicelleştiriciler ilgili dönüşüm katsayılarına uygulanırlar. Nicelleştirilmiş katsayılar bir Huffman kod tablosu kullanılarak kodlanırlar ve böylece bit oranı daha da azaltılmış olur.

Dönüşüm çözümü son derece çok basittir. Önceklikle Huffman çözümü uygulanır. Dönüşüm katsayıları ise

$$\hat{F}(k) = \begin{cases} F(k).\sigma_k^2 & 1 \le k \le K \\ 0 & K < k < L \end{cases} \tag{7.8}$$

ile verilir. Resim bloğu ise

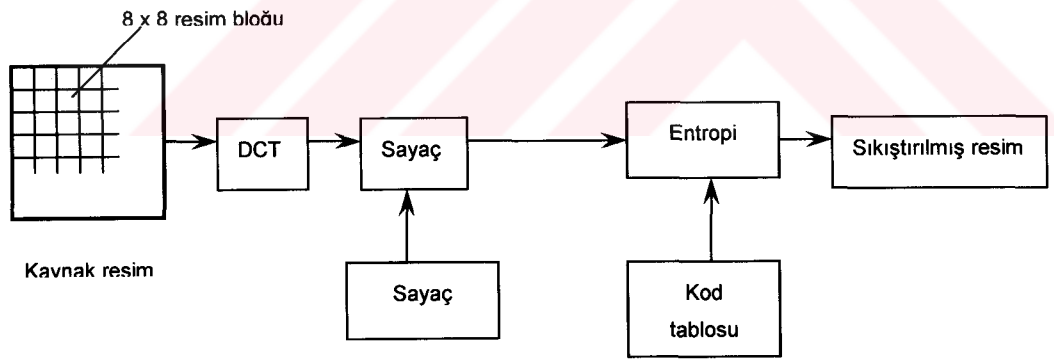
$$\hat{\mathbf{f}}(k) = \mathbf{A}^{-1}\hat{\mathbf{F}} \tag{7.9}$$

ters dönüşümü kullanılarak elde edilir.

Resimlerin dönüşüm kodlamasına ait verilebilecek ilk örnek uluslararası sıkıştırma standardı JPEG'in temelidir. Bu yöntem fotoğraf resmi kodlama komitesi ISO/ IEC/ JTC1/ SC2/ WG10 tarafından hazırlanmıştır. Komite ISO/ IEC'nin bir ek grubu olarak 1990'dan önce de birleşik fotoğraf uzmanları grubu adı altında mevcuttu (JPEG). Bu yüzden sıkıştırma standardı JPEG sıkıştırma olarak bilinmektedir. Bu grup CCITT SGVIII'nın özel bir raportör komitesi ile işbirliği yapmıştır. Bu ortak çalışmada ISO/ IEC kodlama tekniklerini seçer, geliştirir ve test ederken CCITT SGVIII faksimile ve videoteks gibi resim iletimi uygulamalarının kodlama özelliklerini belirlemiştir. Tıbbi, bilimsel ve grafik sanatlar gibi diğer önemli uygulamalara da özel ilgi gösterilmiştir.

JPEG standardı, gri skala ve renkli resim kodlaması için kullanılır, ikili resim kodlaması için uygun değildir. Kodlama ve çözme hakkında özellikler, sıkıştırılmış resim bilgisinin formatını ve sıkıştırma ve çözmenin nasıl yapılacağına dair bilgiyi sağlar. Temel kodlama işleminin yanısıra ilgili başka algoritmalar hakkında da bilgi verir. Tam bir tanımı özellikle buna ayrılmış bir kitabın konusu olabilir. Bu bölümde işleyişinin temel ilkelerini vereceğiz.

JPEG standardı kayıplı ve kayıpsız resim sıkıştırmaları için algoritmalar sağlar. Dört farklı işleyiş şekline bölünebilir. Kayıpsız, sıralı DCT temelli, İlerleyen DCT temelli ve hierarşik olmak üzere. Kayıpsız teknikler Şekil 6.3'te anlatılan öngörülü algoritmaya benzemektedir. En basit kayıplı teknik sıralı DCT temelli kodlamadır ve temel sıralı işlem olarak anılır. Bir çok çeşitli uygulama için yeterlidir. Herhangi bir JPEG kodlaması içerisinde uygulanır. Bu yüzden burada daha detaylı olarak açıklanacaktır. Kodlayıcının akış diagramı Şekil 7.3'te gösterilmiştir.




Şekil 7.3 JPEG temel kodlayıcının basitleştirilmiş diagramı. Pitas, I., (2000)

İlk ön işlem adımı tüm kaynak resme uygulanan seviye kaymasıdır. $B$ resmin gösterilimi için kullanılan bit sayısı olsun. Çoğu durumda 256'lı gri skala kullanılır ve $B = 8$'dir. Bütün resim pikselleri $2^{B-1}$'i çıkartarak işaretli gösterime dönüştürülmüştür. $B = 8$ için seviye kayması 128'dir. JPEG sıkıştırması 8 x 8 bloklar kullanır. Eğer resmin boyutları 8'in katları değilse ihtiyaç duyulan resim blokları eklenir. Eklenen bilgiler çözme sırasında gözardı edilir. DCT

her resim bloğuna uygulanır. Bunun için aşağıdaki dönüşüm tanımı kullanılır:

$$C(k_1,k_2)=\frac{1}{4}v_1(k_1)v_2(k_2)\sum_{n_1=0}^{7}\sum_{n_2=0}^{7}x(n_1,n_2)\cos\frac{(2n_1+1)k_1\pi}{16}\cos\frac{(2n_2+1)k_2\pi}{16} \quad (7.10)$$

Burada $k_1,k_2=0$ ve $v_1(k_1),v_2(k_2)=0$ için $v_1(k_1),v_2(k_2)=1/\sqrt{2}$ 'dir.

İleri DCT hesabından sonra 64 DCT katsayısı düzgün bir nicelleyici tarafından nicellenir :

$$C_q(k_1,k_2)=round(C(k_1,k_2)/Q[k_1][k_2]) \quad (7.11)$$

Burada en yakın tamsayıya yuvarlama yapılmıştır. Böylece nicelleştirme adımı büyüdükçe nicelleştirme hatası da büyür. Nicelleştirici adımları nicelleştirme tablosuna dahildirler $Q[k_1][k_2]$. JPEG içinde varsayılmış herhangi bir nicelleştirme tablosu yoktur ama örnek olarak tipik nicelleştirme tabloları verilmiştir. Bu tarz bir tablo Şekil 7.4'te gösterilmiştir.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 | 11 | 10 | 16 | 24 | 40 | 51 | 61 |
| 12 | 12 | 14 | 19 | 26 | 58 | 60 | 55 |
| 14 | 13 | 16 | 24 | 40 | 57 | 69 | 56 |
| 14 | 17 | 22 | 29 | 51 | 87 | 80 | 62 |
| 18 | 22 | 37 | 56 | 68 | 109 | 103 | 77 |
| 24 | 35 | 55 | 64 | 81 | 104 | 113 | 92 |
| 49 | 64 | 78 | 87 | 103 | 121 | 120 | 101 |
| 72 | 92 | 95 | 98 | 112 | 100 | 103 | 99 |

Şekil 7.4 Resim parlaklığı için 8 x 8'lik bir nicelleştirme tablosu. Pitas, I., (2000)

Nicelleştirilmiş DCT değerleri 11 bit kesimliğinde işaretli iki bütünlerli tamsayılardır(8 bitlik kaynak piksel kesinliği için).

Nicelleştirmeden sonra entropi veya aritmetik kodlama gerçekleştirilir. Nicelleştirilmiş DC katsayısı $C$ (0,0) yüksek düzeyde enerji taşır ve entropi ve aritmetik kodlamaya göre farklı şekilde öncelikli olarak kodlanır. $C_i$ (0,0), $C_{i-1}$ (0,0) değerleri birbirine komşu resim bloklarının DC katsayısını göstersin. Bir tarama işleminin başlangıcında $C_{i-1}$ (0,0) tahmincisi 0'a eşitlenir. $d = C_i$ (0,0) – $C_{i-1}$ (0,0) farkı entropi kodlaması için kullanılır. Nicelleştirilmiş 63

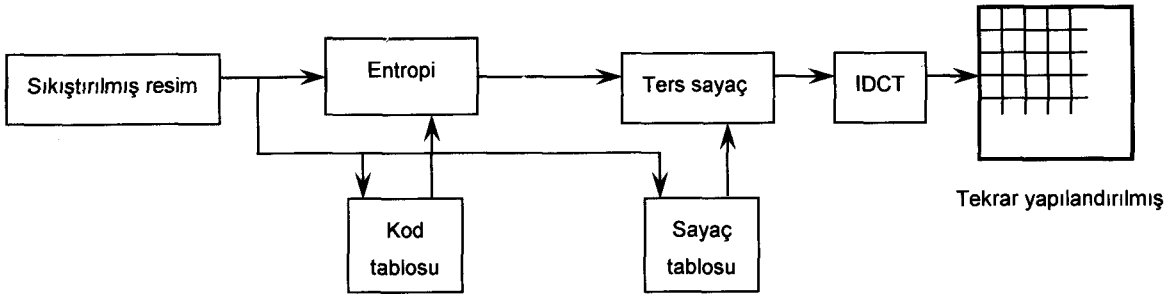
AC, DCT katsayısının geri kalanı tek boyutlu zig zag sıralamasına dönüştürülür. Burada Şekil 7.5'te gösterildiği gibi 8 x 8'lik DCT bloğu taranarak *ZZ* (1),...,*ZZ*(63) elde edilir.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 5 | 6 | 14 | 15 | 27 | 28 |
| 2 | 4 | 7 | 13 | 16 | 26 | 29 | 42 |
| 3 | 8 | 12 | 17 | 25 | 30 | 41 | 43 |
| 9 | 11 | 18 | 24 | 31 | 40 | 44 | 53 |
| 10 | 19 | 23 | 32 | 39 | 45 | 52 | 54 |
| 20 | 22 | 33 | 38 | 46 | 51 | 55 | 60 |
| 21 | 34 | 37 | 47 | 50 | 56 | 59 | 61 |
| 35 | 36 | 48 | 49 | 57 | 58 | 62 | 63 |

Şekil 7.5 8 x 8 DCT katsayı tablosunun zig zag taraması. Pitas, I., (2000)

Nicelleştirilmiş AC katsayılarının çoğu, özellikle yüksek sıklıktakilere denk gelenler, 0'dır. 0'lar kolaylıkla teşhis edilir ve kodlanır. Eğer *ZZ* katarının son bölümü 0 ise bu durum EOB kod kelimesi ile kodlanır.

Ön işleme tabi tutulmuş DCT katsayılarını kodlamak için iki farklı seçenek vardır : Huffman kodlaması ve aritmetik kodlama. Sıralı DCT sıkıştırmasının temeli Huffman kodlamasında kullanılırken, genişletilmiş DCT temelli şemalar Huffman veya aritmetik kodlamayı kullanırlar. AC ve DC, DCT katsayıları kodlaması için farklı Huffman kod tabloları kullanır. Söz dizimi ve oluşturulmaları standardın içinde tanımlanmıştır. Aritmetik kodlama durumda şartlama tablosu özellikleri sağlanır.




Şekil 7.6 JPEG temel çözücüsünün basitleştirilmiş diagramı. Pitas, I., (2000)

JPEG temel çözümleme işlemi Şekil 7.6'da gösterilmiştir. İlk adım Huffman veya aritmetik çözümlemedir. AC DCT katsayı çözümlemesi zig zag kataın $ZZ$ (1),....,$ZZ$(63) elemanlarını üretir. DC katsayı farkı olan $d$ çözülür ve DC terimi olan $C_i$ (0,0) = $C_{i-1}$ (0,0) + $d$'yi üretmekte kullanılır. Nicelleştirme işleminin tersi çözülmüş katsayılarla nicelleştirme tablosu verileri olan $C$ ($k_1$,$k_2$) = $C_q$ ($k_1$,$k_2$)$Q$[$k_1$][$k_2$]'ler çarpılarak oluşturulur. Açıktır ki nicelleştirme ve tersi operasyonlar yeniden oluşturulan resmin kalitesinde kayıplara sebep olurlar. JPEG temel çözüm algoritmasında son aşama ters DCT'dir:

$$x(n_1,n_2)=\frac{1}{4}\sum_{k_1=0}^{7}\sum_{k_2=0}^{7}v_1(k_1)v_2(k_2)C(k_1,k_2)\cos\frac{(2n_1+1)k_1\pi}{16}\cos\frac{(2n_2+1)k_2\pi}{16} \qquad (7.12)$$

ve ayrıca ters kaydırma seviyesidir. 256 gri skalası resimler için seviye kaydırma ters DCT ile elde edilen bütün resim piksellerine 128 değeri ekleyerek oluşturulur.

Şekil 7.7'de bir DCT kodlama örneği gösterilmiştir. Kodlama için 8 x 8'lik blok boyutu kullanılmıştır. Yeniden oluşturulmuş siyah beyaz resim piksel başına yarım bite kadar sıkıştırılmıştır ve Şekil 7.7.b'de gösterilmiştir. Ciddi sıkıştırma oranı bloklama etkisi ile sonuçlanır. Daha yüksek kalite piksel başına bir bitlik sıkıştırma kullanılarak elde edilir.

(a) (b)

Şekil 7.7 (a) Orjinal resim ; (b) JPEG formatında sıkıştırılmış resim. Pitas, I., (2000)

# 8. SONUÇLAR

Bu çalışmada Huffman Kodlaması, Run Length Kodlaması, Değiştirilmiş READ Kodlaması, LZW Sıkıştırılması, Öngörülü Kodlama ve Dönüşüm Resim Kodlaması algoritmarı ile resimler üzerinde hafıza alanını ve iletim zamanını en aza indirgeyecek şekilde sıkıştırma çalışmaları ve analizleri yapılmıştır. Bu yöntemlerde herbir kodlamanın dosya tipine ve kayıplı ya da kayıpsız sıkıştırma durumu olmasına göre performansları farklılıklar arz etmektedir. Kayıpsız Huffman, LZW ve Run-Length kodlamaları genel olarak veri kaybının önemli olduğu yerlerde kullanılmasının daha uygun olduğu anlaşılmış, en yaygın olarak ise LZW sıkıştırmasının en verimli performansla iş gördüğü anlaşılmıştır. Kayıpsız Huffman kodlamasının da veri içerisindeki resim yoğunluk seviyeleri normal dağılımın üzerindeki bilgi katarları için iyi sonuç çıkardığı analiz edilmiştir. Kayıplı sıkıştırmalarda da JPEG sıkıştırma standartları tarafından da benimsenmiş olan dönüşüm resim kodlamasının DCT ile kayıp oranları belirlenerek en büyük verimi sağladığı anlaşılmıştır.

**KAYNAKLAR**

Ahmed, N. , Rao, K.R., (1975) "Orthogonal Transforms for Digital Signal".

Clark, R. J., (1985) "Transform Coding of Images".

Gallagher, R.G., (1968) "Information Theory and Reliable Communications".

Horowitz, E., (1984) "Fundamentals of Computer Algorithms".

Jain, A.K., (1989) "Fundamentals of Digital Image Processing".

Jayant, N.S., Noll, P., (1984) "Digital Coding of Waveforms".

Jayant, N.S., Noll, P., (1984) "Digital Coding of Waveforms".

Knuth, D.E., (1973) "The Art of Computer Programming".

Pitas, I. , (2000) "Digital Image Processing Algorithms and Applications".

Rao , K.R., (1990) "Discrete Cosine Transform Algorithms, Advantages, Application".

**EKLER**

## Ek 1 Huffman C Kodu

```
/*    huffmanendecode.c    */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#ifdef WIN32
#include <io.h>
#include <fcntl.h>
#endif

#define IN_ALPH_LEN      256

struct tree_entry {
        int                     c;
        unsigned long           code;
        int                     bits;
        unsigned long           count;
        struct tree_entry       *left, *right;
};

struct table_entry {
        unsigned long code;
        int             bits;
};

struct heap {
        struct tree_entry       *data[IN_ALPH_LEN];
        int                     entries;
};

int sure_fgetc(FILE *in)
{
        int c = fgetc(in);

        if (c == EOF)
        {
                fputs("Beklemedik dosya sonu !\n", stderr);
                exit(4);
        }

        return c;
}

void output(int c, FILE *out, struct table_entry *table)
{
        static int bits;
        static unsigned long chunk;
```

```
        if (c >= 0)
        {
                chunk <<= table[c].bits;
                chunk |= table[c].code;
                bits += table[c].bits;

                while (bits >= 8)
                {
                        bits -= 8;
                        fputc((chunk >> bits) & 0xff, out);
                }
        }
        else
        {
                chunk <<= (8 - bits);
                fputc(chunk & 0xff, out);
        }
}

void add_heap_entry(struct heap *h, struct tree_entry *n)
{
        int i;

        for (i = h->entries; i && h->data[i - 1]->count < n->count; i--)
                h->data[i] = h->data[i - 1];

        h->data[i] = n;
        h->entries++;
}

struct tree_entry *get_heap_entry (struct heap *h)
{
        if (!h->entries)
        {
                fprintf(stderr, " Bos dosyaya huffman algoritmasini uyguluyorsunuz!\n");
                exit(7);
        }
        return h->data[--h->entries];
}

struct tree_entry *build_tree(unsigned long *stats)
{
        int i;
        struct heap h;
        struct tree_entry *n;
        unsigned long max_count = 0;

        h.entries = 0;

        for (i = 0; i < IN_ALPH_LEN; i++)
```

```
        {
                if (!stats[i])
                        continue;
                if (!(n = malloc(sizeof(*n))))
                {
                        perror("malloc() failed in build_tree");
                        return NULL;
                }
                n->c = i;
                n->left = NULL;
                n->right = NULL;
                n->count = stats[i];
                if (n->count > max_count)
                        max_count = n->count;
                add_heap_entry(&h, n);
        }

        while(h.entries > 1)
        {
                if (!(n = malloc(sizeof(*n))))
                {
                        perror("malloc() failed in build_tree()");
                        return NULL;
                }
                n->left = get_heap_entry(&h);
                n->right = get_heap_entry(&h);
                n->count = n->left->count + n->right->count;
                add_heap_entry(&h, n);
        }

        n = get_heap_entry(&h);
        n->count = max_count;
        return n;
}

void attach_codes(struct tree_entry *tree, unsigned long code, int deepth)
{
        if (!tree->right && !tree->left)
        {
                tree->bits = deepth;
                tree->code = code;
                return;
        }

        if (tree->left)
                attach_codes(tree->left, code << 1, deepth + 1);

        if (tree->right)
                attach_codes(tree->right, code << 1 | 1, deepth + 1);
}
```

```
void build_table(struct tree_entry *tree, struct table_entry *table)
{
	if (!tree->left && !tree->right)
	{
		table[tree->c].code = tree->code;
		table[tree->c].bits = tree->bits;
		return;
	}

	if (tree->left)
		build_table(tree->left, table);

	if (tree->right)
		build_table(tree->right, table);
}

void write_header(unsigned long *stats, unsigned long bytes,
	unsigned long max_count, FILE *out)
{
	int i, bits = (max_count < (1 << 16))?16:32;

	fputc(bits, out);

	for (i = 0; i < IN_ALPH_LEN; i++)
	{
		if (stats[i])
		{
			fputc(i, out);
			if (bits == 32)
			{
				fputc(stats[i] >> 24 & 0xff, out);
				fputc(stats[i] >> 16 & 0xff, out);
			}
			fputc(stats[i] >> 8 & 0xff, out);
			fputc(stats[i] & 0xff, out);
		}
	}

	fputc(0, out);

	fputc(bytes >> 24 & 0xff, out);
	fputc(bytes >> 16 & 0xff, out);
	fputc(bytes >> 8 & 0xff, out);
	fputc(bytes & 0xff, out);
}

int encode(FILE *in, FILE *out)
{
	unsigned long stats[IN_ALPH_LEN] = {0}, bytes = 0;
	struct tree_entry *tree;
	struct table_entry table[IN_ALPH_LEN] = {{0, 0}};
```

```
        FILE *tmp = tmpfile();
        int c;
        printf("Sikistirma islemi yapiliyor...\n");
        if (!tmp)
        {
                fprintf(stderr, "Gecici dosya yaritilamadi : %s\n",
                        strerror(errno));
                return 5;
        }

        while ( (c = fgetc(in)) != EOF)
        {
                fputc(c, tmp);
                stats[c]++;
                bytes++;
        }
        rewind(tmp);

        if (!(tree = build_tree(stats)))
                return 9;

        attach_codes(tree, 0, 0);
        build_table(tree, table);
        write_header(stats, bytes, tree->count, out);

        while ( (c = fgetc(tmp)) != EOF)
                output(c, out, table);
        output(-1, out, table);

        fclose(tmp);

        return 0;
}

unsigned long read_header(FILE *in, unsigned long *stats)
{
        int bits, flag = 0, c;
        unsigned long bytes;

        if (((bits = sure_fgetc(in)) != 16) && (bits != 32))
        {
                fprintf(stderr, " 16 ya da 32 bit huffman?\n");
                return -1UL;
        }

        for(;;)
        {
                c = sure_fgetc(in);
                if (flag && !c)
                        break;
                else
```

```
                flag = 1;
            if (bits == 32)
            {
                stats[c] = sure_fgetc(in) << 24;
                stats[c] |= sure_fgetc(in) << 16;
            }
            stats[c] |= sure_fgetc(in) << 8;
            stats[c] |= sure_fgetc(in);
        }

        bytes = sure_fgetc(in) << 24;
        bytes |= sure_fgetc(in) << 16;
        bytes |= sure_fgetc(in) << 8;
        bytes |= sure_fgetc(in);

        return bytes;
}

struct tree_entry *get_tree_entry(struct tree_entry *tree,
        unsigned long chunk, int bits, int deepth)
{
        if (deepth > bits)
                return NULL;

        if (!tree->left && !tree->right)
                return tree;

        if (tree->left && !((chunk >> (bits - deepth - 1)) & 1))
                return get_tree_entry(tree->left, chunk, bits, deepth + 1);

        if (tree->right && ((chunk >> (bits - deepth - 1)) & 1))
                return get_tree_entry(tree->right, chunk, bits, deepth + 1);

        return NULL;
}

int print_code(int c, unsigned long bytes, struct tree_entry *tree, FILE *out)
{
        static int bits;
        static unsigned long chunk;
        int decoded_bytes = 0;
        struct tree_entry *e;

        chunk <<= 8;
        chunk |= c;
        bits += 8;
        if (bits > 32)
        {
                fprintf(stderr, "Cod basina 24 bitten fazlasi işlenemez!\n");
                exit(8);
        }
```

```
        while (bytes-- && (e = get_tree_entry(tree, chunk, bits, 0)))
        {
                decoded_bytes++;
                fputc(e->c, out);
                bits -= e->bits;
        }

        return decoded_bytes;
}

int decode(FILE *in, FILE *out)
{
        unsigned long bytes, stats[IN_ALPH_LEN] = {0};
        struct tree_entry *tree;
        int c;
        printf("Cozme islemi yapiliyor...\n");
        if ((bytes = read_header(in, stats)) == -1UL)
                return 6;

        if (!(tree = build_tree(stats)))
                return 9;

        attach_codes(tree, 0, 0);

        while(bytes && ((c = fgetc(in)) != EOF))
                bytes -= print_code(c, bytes, tree, out);

        return 0;
}

int main(int argc, char *argv[])
{
        char *infile = NULL, *outfile = NULL;
        FILE *in, *out;
        int action = 0, i;
        char *prg = strrchr(argv[0], '/');

#ifdef WIN32
        setmode(0, O_BINARY);
        setmode(1, O_BINARY);
#endif

        if (!prg++)
                prg = argv[0];

        if (!strcmp(prg, "huff"))
                action = 1;
        if (!strcmp(prg, "unhuff"))
                action = 2;
```

```
for (i = 1; i < argc; i++)
{
        if (!strcmp(argv[i], "-e"))
        {
                action = 1;
                continue;
        }

        if (!strcmp(argv[i], "-d"))
        {
                action = 2;
                continue;
        }

        if (!infile)
        {
                infile = argv[i];
                continue;
        }

        if (!outfile)
        {
                outfile = argv[i];
                continue;
        }


        fprintf(stderr, "\"%s\" anlasilamadi \n", argv[i]);
        action = 0;
        break;
}

if (!action)
{
        fprintf(stderr, "kullanim : %s <-e|-d> [girdi dosyasi] [cikti dosyasi]\n",
                argv[0]);
        return 1;
}

if (infile)
{
        if (!(in = fopen(infile, "rb")))
        {
                fprintf(stderr, " %s dosyasi acilamadi : %s\n",
                        infile, strerror(errno));
                return 2;
        }
}
else
        in = stdin;
```

```
        if (outfile)
        {
                if (!(out = fopen(outfile, "wb")))
                {
                        fprintf(stderr, " %s dosyasi acilamadi : %s\n",
                                outfile, strerror(errno));
                        return 3;
                }
        }
        else
                out = stdout;

        if (action == 1)
                return encode(in, out);
        else
                return decode(in, out);
}
```

## Ek 2 Run-Length C Kodu

```
/*   rlendecode.c   */

#include <stdio.h>
#include <string.h>
#include <dir.h>
#include <errno.h>
#define MAX_LEN   (0x7fff)
#define MAX_RUN_HEADER (0xffff)
#define MAX_SEQ_HEADER (0x7fff)
#define RUN (0x8000)
#define SEQ (0x0000)

int main(int argc, char *argv[])
{
        char *infile = NULL, *outfile = NULL;
        FILE *in, *out;
        int action = 0, i;
        char *prg = strrchr(argv[0], '/');

#ifdef WIN32
        setmode(0, O_BINARY);
        setmode(1, O_BINARY);
#endif

        if (!prg++)
                prg = argv[0];

        if (!strcmp(prg, "rle"))
                action = 1;
        if (!strcmp(prg, "rld"))
                action = 2;

        for (i = 1; i < argc; i++)
        {
                if (!strcmp(argv[i], "-e"))
                {
                        action = 1;
                        continue;
                }

                if (!strcmp(argv[i], "-d"))
                {
                        action = 2;
                        continue;
                }

                if (!infile)
                {
                        infile = argv[i];
```

```
                continue;
        }

        if (!outfile)
        {
                outfile = argv[i];
                continue;
        }


        fprintf(stderr, "\"%s\" anlasilamadi \n", argv[i]);
        action = 0;
        break;
}

if (!action)
{
        fprintf(stderr, "kullanim : %s <-e|-d> [girdi dosyasi] [cikti dosyasi]\n",
                argv[0]);
        return 1;
}

if (infile)
{
        if (!(in = fopen(infile, "rb")))
        {
                fprintf(stderr, " %s dosyasi acilamadi : %s\n",
                        infile, strerror(errno));
                return 2;
        }
}
else
        in = stdin;

if (outfile)
{
        if (!(out = fopen(outfile, "wb")))
        {
                fprintf(stderr, " %s dosyasi acilamadi : %s\n",
                        outfile, strerror(errno));
                return 3;
        }
}
else
        out = stdout;

if (action == 1)
        return encode(in, out);
else
        return decode(in, out);
}
```

```
int decode(FILE *infile, FILE *outfile)
{
   register int byte;
   register unsigned short i;
   register unsigned short length;
   int packet_hdr;

   char orig_filename[14];
   char *infile_name;
   char scratch_space[134];

        printf("Cozme islemi yapiliyor...\n");

        while (!feof(infile))
   {
      packet_hdr = fgetc(infile);

      if (feof(infile))
         continue;

      packet_hdr |= (((short)fgetc(infile)) << 8) ;

      if (feof(infile))
         continue;


      length = MAX_LEN & packet_hdr;

      if (packet_hdr & RUN)
      {
         byte = fgetc(infile);

         for (i = 0; i < length; i++)
            fputc(byte, outfile);
      }

      else

         for (i = 0; i < length; i++)
            fputc(fgetc(infile), outfile);

   }
   fclose(infile);
   fclose(outfile);

        return 0;
}
```

```
int encode(FILE *infile, FILE *outfile)
{
   register int cur_char;
   register unsigned int i;
   register unsigned short run_len = 0;
   int run_char;
   unsigned int j;
   unsigned short seq_len=0;

   char seq[MAX_LEN];
   char scratch_space[256];

   char orig_name[MAXFILE+MAXEXT];
   char drive[MAXDRIVE],
       dir[MAXDIR],
       filename[MAXFILE],
       ext[MAXEXT];
   char outfile_name[MAXFILE+MAXEXT];

   unsigned long  orig_total;
   unsigned long  rle_total;
   char *infile_name;

         printf("Sikistirma islemi yapiliyor...\n");

         while (!feof(infile))
   {
      cur_char = fgetc(infile);

      if (feof(infile))
         continue;


      if (seq_len ==0)
      {
         if (run_len == 0)
         {
            run_char = cur_char;
            ++run_len;
            continue;
         }

         if (run_char == cur_char)
            if (++run_len == MAX_LEN)
            {
               fputc((int)MAX_RUN_HEADER & (0xff), outfile);
               fputc((int)MAX_RUN_HEADER >> 8, outfile);
               fputc((int) run_char, outfile);

               run_len = 0;
```

```
        continue;
      }


   if (run_len > 3)


   {
      fputc((int)(run_len & 0xff), outfile);
      fputc((int)(RUN | run_len) >> 8, outfile);
      fputc((int)run_char, outfile);

      run_len = 1;
      run_char  = cur_char;
      continue;
   }

   for (j = 0; j < run_len; j++);
   {
      seq[seq_len] = run_char;
      ++seq_len;
      if (seq_len == MAX_LEN)
      {
         fputc((int)MAX_SEQ_HEADER & 0xff, outfile);
         fputc((int)(MAX_SEQ_HEADER >> 8) , outfile);

         for (i = 0; i < seq_len; i++)
            fputc((int)seq[i], outfile);

         seq_len = 0;
      }
   }

   run_len = 0;

   seq[seq_len++] = cur_char;
   if (seq_len == MAX_LEN)
   {
      fputc((int)MAX_SEQ_HEADER & 0xff, outfile);
      fputc((int)(MAX_SEQ_HEADER >> 8) , outfile);

      for (i = 0; i < seq_len; i++)
         fputc((int)seq[i], outfile);

      seq_len = 0;
   }
}
else
{
   if (run_len != 0)
```

```
{
   if (cur_char == run_char )
   {
      ++run_len;
      if (run_len == MAX_LEN)
      {
         fputc((int)seq_len & 0xff, outfile);
         fputc((int)((SEQ | seq_len) >> 8), outfile);

         for (i = 0; i < seq_len; i++)
            fputc((int)seq[i], outfile);


         fputc((int)(run_len & 0xff), outfile);
         fputc((int)(RUN | run_len) >> 8, outfile);

         fputc((int)run_char, outfile);

         seq_len = run_len = 0;

      }
      continue;

   }

   fputc((int)seq_len & 0xff, outfile);
   fputc((int)((SEQ | seq_len) >> 8), outfile);

   for (i = 0; i < seq_len; i++)
      fputc((int)seq[i], outfile);


   fputc((int)(run_len & 0xff), outfile);
   fputc((int)(RUN | run_len) >> 8, outfile);

   fputc((int)run_char, outfile);

   seq_len = 0;
   run_len = 1;
   run_char = cur_char;

   continue;

}


if (seq[seq_len - 1] == cur_char)
{
   run_char = cur_char;
   run_len = 2;
   --seq_len;
```

```
            continue;
        }


        seq[seq_len++] = cur_char;

        if (seq_len == MAX_LEN)
        {
            fputc((int)MAX_SEQ_HEADER & 0xff, outfile);
            fputc((int)(MAX_SEQ_HEADER >> 8) , outfile);

            for (i = 0; i < MAX_LEN; i++)
                fputc((int)seq[i], outfile);

            seq_len = 0;
        }

    }

}


if (seq_len != 0)
{
    fputc((int)seq_len & 0xff, outfile);
    fputc((int)((SEQ | seq_len) >> 8), outfile);

    for (i = 0; i < seq_len; i++)
        fputc((int)seq[i], outfile);
}

if (run_len != 0)
{
    fputc((int)(run_len & 0xff), outfile);
    fputc((int)(RUN | run_len) >> 8, outfile);

    fputc((int)run_char, outfile);
}

fclose(infile);
fclose (outfile);

return 0;

}
```

## Ek 3 LZW C Kodu

```
/*   lzwendecode.c   */

#include <stdio.h>
#include <errno.h>
#define BITS 12
#define HASHING_SHIFT BITS-8
#define MAX_VALUE (1 << BITS) - 1
#define MAX_CODE MAX_VALUE - 1

#if BITS == 14
 #define TABLE_SIZE 18041
#endif
#if BITS == 13
 #define TABLE_SIZE 9029
#endif
#if BITS <= 12
 #define TABLE_SIZE 5021
#endif

void *malloc();

int *code_value;
unsigned int *prefix_code;
unsigned char *append_character;
unsigned char decode_stack[4000];


int main(int argc, char *argv[])
{
        char *infile = NULL, *outfile = NULL;
        FILE *in, *out, *lzw;
        int action = 0, i;
        char *prg = strrchr(argv[0], '/');

        code_value=malloc(TABLE_SIZE*sizeof(unsigned int));
        prefix_code=malloc(TABLE_SIZE*sizeof(unsigned int));
        append_character=malloc(TABLE_SIZE*sizeof(unsigned char));

        if (code_value==NULL || prefix_code==NULL || append_character==NULL)
        {
                printf("Fatal error olustu, ayrilmis tablo alani!\n");
                exit();
        }

#ifdef WIN32
        setmode(0, O_BINARY);
        setmode(1, O_BINARY);
#endif
```

```
if (!prg++)
        prg = argv[0];

if (!strcmp(prg, "lzw"))
        action = 1;
if (!strcmp(prg, "unlzw"))
        action = 2;

for (i = 1; i < argc; i++)
{
        if (!strcmp(argv[i], "-e"))
        {
                action = 1;
                continue;
        }

        if (!strcmp(argv[i], "-d"))
        {
                action = 2;
                continue;
        }

        if (!infile)
        {
                infile = argv[i];
                continue;
        }

        if (!outfile)
        {
                outfile = argv[i];
                continue;
        }


        fprintf(stderr, "\"%s\" anlasilamadi \n", argv[i]);
        action = 0;
        break;
}

if (!action)
{
        fprintf(stderr, "kullanim : %s <-e|-d> [girdi dosyasi] [cikti dosyasi]\n",
                argv[0]);
        return 1;
}

if (infile)
{
        if (!(in = fopen(infile, "rb")))
        {
```

```
                fprintf(stderr, " %s dosyasi acilamadi : %s\n",
                        infile, strerror(errno));
                return 2;
            }
        }
        else
            in = stdin;

        if (outfile)
        {
            if (!(out = fopen(outfile, "wb")))
            {
                fprintf(stderr, " %s dosyasi acilamadi : %s\n",
                        outfile, strerror(errno));
                return 3;
            }
        }
        else
            out = stdout;

        if (action == 1) {
            compress(in,out);
            fclose(in);
            fclose(out);
            free(code_value);
        }
        else {
            expand(in,out);
            fclose(in);
            fclose(out);
            free(prefix_code);
            free(append_character);
        }
}


compress(FILE *input,FILE *output)
{
        unsigned int next_code;
        unsigned int character;
        unsigned int string_code;
        unsigned int index;
        int i;

        next_code=256;
        for (i=0;i<TABLE_SIZE;i++)
            code_value[i]=-1;

        i=0;
        printf("Sikistirma islemi yapiliyor...\n");
        string_code=getc(input);
```

```
        while ((character=getc(input)) != (unsigned)EOF)
        {
                if (++i==1000)
                {
                        i=0;
          }
                index=find_match(string_code,character);
                if (code_value[index] != -1)
                        string_code=code_value[index];
                else
                {
                        if (next_code <= MAX_CODE)
                        {
                                code_value[index]=next_code++;
                                prefix_code[index]=string_code;
                                append_character[index]=character;
                        }
                        output_code(output,string_code);
                        string_code=character;
                }
        }
        output_code(output,string_code);
        output_code(output,MAX_VALUE);
        output_code(output,0);
}


find_match(int hash_prefix,unsigned int hash_character)
{
        int index;
        int offset;

        index = (hash_character << HASHING_SHIFT) ^ hash_prefix;
        if (index == 0)
          offset = 1;
        else
          offset = TABLE_SIZE - index;
        while (1)
        {
                if (code_value[index] == -1)
                        return(index);
          if (prefix_code[index] == hash_prefix && append_character[index] ==
hash_character)
                        return(index);
        index -= offset;
   if (index < 0)
                index += TABLE_SIZE;
        }
}

expand(FILE *input,FILE *output)
```

```
{
        unsigned int next_code;
        unsigned int new_code;
        unsigned int old_code;
        int character;
        int counter;
        unsigned char *string;
        char *decode_string(unsigned char *buffer,unsigned int code);

        next_code=256;
        counter=0;
        printf("Cozme islemi yapiliyor...\n");

        old_code=input_code(input);
        character=old_code;
        putc(old_code,output);
        while ((new_code=input_code(input)) != (MAX_VALUE))
        {
          if (++counter==1000)
                {
                        counter=0;
                }

                if (new_code>=next_code)
                {
                        *decode_stack=character;
                        string=decode_string(decode_stack+1,old_code);
                }
                else
                        string=decode_string(decode_stack,new_code);
                        character=*string;
                        while (string >= decode_stack)
                                putc(*string--,output);
                        if (next_code <= MAX_CODE)
                        {
                                prefix_code[next_code]=old_code;
                                append_character[next_code]=character;
                                next_code++;
                        }
                        old_code=new_code;
                }
}


char *decode_string(unsigned char *buffer,unsigned int code)
{
        int i;

        i=0;
        while (code > 255)
        {
```

```
                *buffer++ = append_character[code];
                code=prefix_code[code];
                if (i++>=4094)
                {
                        printf("Codun cozulmesi sirasinda hata olustu!\n");
                        exit();
                }
        }
        *buffer=code;
        return(buffer);
}

input_code(FILE *input)
{
        unsigned int return_value;
        static int input_bit_count=0;
        static unsigned long input_bit_buffer=0L;

        while (input_bit_count <= 24)
        {
                input_bit_buffer |= (unsigned long) getc(input) << (24-input_bit_count);
                input_bit_count += 8;
        }
        return_value=input_bit_buffer >> (32-BITS);
        input_bit_buffer <<= BITS;
        input_bit_count -= BITS;
        return(return_value);
}

output_code(FILE *output,unsigned int code)
{
        static int output_bit_count=0;
        static unsigned long output_bit_buffer=0L;

        output_bit_buffer |= (unsigned long) code << (32-BITS-output_bit_count);
        output_bit_count += BITS;
        while (output_bit_count >= 8)
        {
                putc(output_bit_buffer >> 24,output);
                output_bit_buffer <<= 8;
                output_bit_count -= 8;
        }
}
```

**ÖZGEÇMİŞ**

| | | |
|---|---|---|
| Doğum tarihi | 11.10.1976 | |
| Doğum yeri | İstanbul | |
| Lise | 1991-1995 | Deniz Lisesi Komutanlığı |
| Lisans | 1996-2000 | İstanbul Teknik Üniversitesi Fen Fak. Matematik Mühendisliği Bölümü |
| Yüksek Lisans | 2000-2003 | Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Matematik Müh. Anabilim Dalı |

**Çalıştığı kurum(lar)**

2001-Devam ediyor Treda Bilişim Teknolojileri A.Ş.