

BMG 1.Yarıyıl Güz Projesi

yazar

Muhammed Emin Pehlivan

Bu slaytta neler var?

- 0-)Metin, resim ve ses verilerinin bit düzeyinde temsili
- 1-)Veri sıkıştırılmaya giriş
- 2-)Temel sıkıştırma algoritmaları
- 3-)Kapanış

0.0

Metnin Bit Düzeyine Temsili

Bilgisayarların bir ve sıfırlarla çalıştığını biliyoruz ancak bu bir ve sıfırların bizim dilimizde bir karşılığı yok, her millet kendine ait bir alfabe ile donatıldı haliyle makine ile doğrudan konuşamayız. Makine ile konuşabilmek için girdiğimiz harf, sayı veya semboller makinenin anlayacağı sıfırlar ve birlere çevirmemiz, ayrıca bu farklı dizilimdeki bir ve sıfırları öğretmemiz ve gelen çıktıları aynı şekilde geri insan diline çevirmemiz lazım. Bu da bizi bu işin başlangıcına yani 1960'lara götürür.

1960'larda ASCII tablosu Robert W. Bemer liderliğinde geliştirildi, ilk zamanlar 7 bitten oluşan tabloda 128'e kadar karakter depolamak mümkünü. Ancak bunun bir eksikliği vardı, ilk nesil ASCII tablosunda ingilizce harici kelimeler için yer yoktu ve bu yüzden ISO 8859-1, ISO 8859-2 ve Windows-1252 gibi ASCII tablosunun geliştirilmiş varyasyonları ortaya çıktı.

Ancak zamanla bu da yeterli olmadı, çünkü tüm dünya dillerini kapsamıyordu bu yüzden daha gelişmiş ve evrensel olarak standartlaşması gereken bir karakter tablosu gerekliydi. İlk olarak 1980'lerde bunun üzerine düşünlümeye, yıl 1991'e geldiğinde ise Apple, Microsoft, IBM gibi büyük teknoloji şirketleri bir araya gelmesi sonucu Unicode oluşturuldu ve uluslararası standart olan ISO/IEC 10646 ile senkronize edilerek küresel kabul gördü. Unicode, her karaktere benzersiz kod oluşturması sayesinde bugün latin harflerinden çince ideo-graflara, matematiksel sembollerden emojilere kadar onbinlerce karakter tek bir standart üzerinde birleşmiş oldu.

Karakter	ASCII	Karakter	ASCII	Karakter	ASCII	Karakter	ASCII
@	64	P	80	.	96	p	112
A	65	Q	81	a	97	q	113
B	66	R	82	b	98	r	114
C	67	S	83	c	99	s	115
D	68	T	84	d	100	t	116
E	69	U	85	e	101	u	117
F	70	V	86	f	102	v	118
G	71	W	87	g	103	w	119
H	72	X	88	h	104	x	120
I	73	Y	89	i	105	y	121
J	74	Z	90	j	106	z	122
K	75	[91	k	107	{	123
L	76	\	92	l	108		124
M	77]	93	m	109	}	125
N	78	^	94	n	110	~	126
O	79	-	95	o	111		

ASCII tablodaki ingiliz alfabetesi

Peki bu karakter kodlaması nasıl yapılıyor?

0.1 Karakter Kodlaması nedir?

Karakter kodlaması aslında alfabeteki harhangi bir harf, yada bir sayı, yada bir matematiksel simbolün bayt düzeyindeki karşılığına denir, ilk çıkan ASCII karakter tablosunda karakter 1'den 128'e kadar karakterlerin bayt düzeyinde kaydedilmesini sağlıyordu.

Örneğin A karakteri ASCII tablodaki değeri 65 yani 1000001 olarak gösterilirken (simbolü 40 yani 0101000 ile gösterilir. Tabii ki bunların daha kolay gösterimi olan oktal ve hexadecimal gösterimler de bulunur, ancak hepsi temelde binary versiyonu daha kolay gösterecektir.

Örneğin 5 sayısının binary gösterimi 0110101 iken oktal karşılığı 065, hexadecimal karşılığı ise 35'tir. Burada oktal yerine sadece binary gösterim detaylı anlatılacaktır.

0.2 Binary Gösterim ve Unicode

Binary gösterim yani 1010111 şeklinde yapılan gösterimin nasıl 87'ye eşit olduğunu anlamak için eski uçaklardaki irtifayı gösteren odometre sayacına bakalım:



Örneğin AB irtifası olan bir uçak düşünelim. Başlangıçta irtifa sıfırken A ve B sıfır değerini gösterir, uçak yükseldikçe B değeri 9'a kadar çıkar A değeri ise ancak B değeri 9 olduktan sonra artmaya başlar ve bu değer A ve B değerleri 9'a eşit olana kadar sürer.

Binary gösterimde de aslında bu mantık ile benzer olarak çalışır ancak uçağın sayacına göre A veya B değeri sadece 1 veya olabilmektedir, ayrıca A ve B değeri bildiğimiz onluk sisteme göre çalışır yani her basamak onun katlarına göre şekillenir. Binary gösterim ise ikinin katlarına

göre sayı değeri artar, aslında 7 bit 128 karakter olayı da buradan gelmektedir. Hatta dikkat ettiyseniz günümüzdeki GB ve MB depolama birimleri 10'un katlarına göre değil, ikinin katlarına göredir.

Unicode için baktığımızda ise ASCII tablosuna göre biraz daha karışiktır, çünkü 1 ila 4 bayt aralığında karakterler saklandığı için farklı UTF gösterimlerine sahiptir, her ne kadar hepsi hexadecimal gösterime sahip olsa da farklı kullanım alanlarına sahiptir. Örneğin utf-8 1 ila 4 bayt aralığında değişken uzunluğa sahip, ASCII ile uyumlu ve WEB'de yaygın olarak kullanılmaktadır. utf-16 ise 2 ila 4 bayt aralığında ve genelde windows ile java'da kullanılır. utf-32 gösterimi de bulunur ancak uzun yer kapladığı için pek tercih edilmez. Bu gösterimlerin farkını anlamak için bir örnek verelim. ç harfi utf-8 ile C3A7, utf-16 ile 00E7, utf-32 olarak ise 000000E7 olarak gösterilebilir.

0.3 Siyah-Beyaz Görüntülerin Bit Temsili

Görüntüleri doğrudan bir bilgisayara anlatamayız. Hatta bilgisayarlar görüntünün de ne olduğunu bilmezler. Bilgisayarlar ancak sıfır ve birleri anlayabilir. Dolayısıyla bir görüntüyü digitalleştirmenin yolu bit ve bayt- lardan geçiyor. Bunun en ilkel yöntemi ise siyah beyaz piksellerdi. İlk nesil görüntüler için sadece iki renk mevcuttu, siyah ve beyaz. 0 beyazı, 1 ise siyahı temsil eder. Her piksel bir yada daha fazla bit koleksiyonundan oluşuyor, bu pikseller ise satır ve sütun olmak üzere 2 boyutlu bir ekran oluşturarak bildiğimiz görüntüyü oluşturuyordu. Biz buna bit haritası (bitmap) diyoruz. Bu siyah beyaz görüntüler harfler veya sayılar için yeterliydi ancak bir görüntüde ışık, gölge, yumuşak geçişleri yoktu çünkü o zamanlarda(1950) bu derece detaylı görüntü verisi işleyecek işlem gücü yoktu. Ancak yıl 1970'lere gelindiğinde hem işlemciler ve bellekler yükseldi, hem de veri depolama kapasitesi arttı. Bu sayede gri tonlama ortaya çıktı. Gri tonlama artık bir piksel için 8 bit uzunlığında 256 farklı siyah-beyaz aralığındaki renkleri sunuyordu.

0.4 Renkli Görüntünün Ortaya Çıkışı(RGB,RGBA)

Renkli görüntülerin ortaya çıkışını anlamak için önce gözün bu renkleri nasıl algıladığını anlamamız gerekecek. Retinada iki tip hücre çeşidi vardır. Bunlar çubuk ve koni hücreleridir. Çubuk hücreleri daha çok siyah beyaz görüntü ve cisimlerin şekil yapısını anlamak için gelişmişken, koni hücreleri renkleri, parlaklıği ve keskin kenarları algılamak gelişmiştir. Işığında 3 temel renkten oluşması gibi gözümden de bu 3 renk olan kırmızı mavi ve yeşil renklere duyarlı reseptör hücreler bulunur. Aslında renkli görüntü dediğimiz RGB bu üç rengin farklı oranlarda birleşmesi sonucu istenen renk bize pikseller ile gözüümüze gelir. Her piksel 3 bayttan oluşur ve her bayt bir temel rengi temsil eder. Her renk içinde pikselde bir led bulunur. Bu sayede her led için 256 farklı renk tonu kullanılarak istenen renk ekrana yansıtılır.

Her ne kadar JPEG içinde olsa da burada da YCbCr üzerine得分mekte fayda var. İnsan gözü parlaklığa renklere oranla daha yüksek bir hassasiyeti bulunur.

İşte YCbCr insan gözünün bu özelliğini kullanarak görüntüde sıkıştırma yapar. Buradaki Y değeri parlaklığı, Cb değeri mavi renk farkını, Cr değeri ise kırmızı renk farkıdır. RGB 3 renk kanalını yüksek çözünürlükte kullanırken YCbCr sadece parlaklığı yüksek çözünürlükte alır. Bu sayede biz aynı görüntü kalitesiyile daha az yer kaplayacak şekilde videoları depolar



En sonda sadece siyah beyaz, ortada gri tonları ve en sağda da rgb versiyonunu içeren bir pilot görseli

RGBA ise RGB ile neredeyse aynıdır. Tek fark renklerin haricde saydamlık için de ayrı bir kanala sahip olmasıdır. Bu yönüyle RGB 3 bayt kullanıyorsa RGBA 4 bayt kullanır. Genellikle PNG ikonlar UI/arayüz elemanları oyunlardaki sprite'lar ,transparan overlay efektleri, web grafiklerinde, 2D oyun motorlarında (Unity, Godot), video düzenleme yazılımlarında (green screen, compositing efektleri) kullanılır.

0.5 Sesin Sayısallaştırılması

Sesin dijital temsilini anlayabilmek için önce günümüze kadar seslerin ne tür yöntemlerle depolandığını kısaca anlamamız gereklidir. Öncelikle ses sürekli(analog) bir basınç dalgasıdır. Yani kesintisiz ve sonsuz bir değere sahiptir. 1877'de Thomas Edison'un icat ettiği fonograf ses dalgalarındaki enerjinin bir diyaframı hareket ettirmesiyle balmumuna fiziksel olarak çizikler bırakıyor, bu sayede sesin fiziksel olarak depolanması sağlanıyordu. Daha sonra gramafon plakları ve manyetik bantlar gibi yeni yöntemlerde çıktı. Ancak hepsinin sorunu ayniydi, Analog olmaları.

Analog depolamada ses doğrudan dalganın fiziksel biçiminde saklanır, bu başta normal gibi gözüksede pratikte çok ciddi sınırlamaları beraberinde getirir. Analogun sürekli olması elektriksel parazitler, mekanik titreşimler, manyetik alan, ısı ve zamanla oluşan fiziksel aşınma doğrudan sinyalin içine girer. Bu ayrı yapı gibi düşünülmemelidir, bu yüzden bu tür pürüzlerin tamamen temizlenmesi mümkün değildir. Sesin her kopyalanmasında ses orjinalinden daha farklı olacaktır. Plaklarda fiziki deformasyon, manyetik bantlarda manyetik alanın zayıflaması, kasetlerin nem ve sıcaklıklardan etkilenmesi sesin analog olarak depolanmasını olumsuz etkilemiştir. Ayrıca analog sesle çalışmanın da zor olması(hassas değil, yavaş, geri alma yok, hatalar kalıcı vb.) ve depolamanın verimsiz olması artık sesin depolanmasında yeni bir yöntemin gerekliliği konusunda bizi arayışa sürüklemiştir.

Bilgisayarlar kesikli değerler(0 ve 1'ler) ile çalışır, haliyle analog ses kaydı için uygun değildi. Çözüm olarak ise sesleri sayıya dönüştürme fikri, sesin dijitalleştirilmesi noktasındaki ilk kıvılcım işte burada atılıyor.

~~~~~ Bu bir analog ses örneğidir. Bizim yaptığımız ise sesin saniyede binlerce defa "bu andaki sayısal değeri nedir" sorusunu sormak. Bu işleme örnekleme denir. Peki biz örnekleme hızını neye göre karar vereceğiz, işte bu noktada Nyquist teoremi bize şunu söyler: "Bir analog sinyalindeki en yüksek frekansın en az iki katı hızla örnekleme yapılmazsa sinyal doğru temsil edilemez". Örneğin duyulması istenen sesin maksimum frekansı 20 kHz ise minimum örnekleme hızı 40 kHz olmalıdır. Aslında bu yüzden de CD'lerde ses kalitesi 44.1 kHz olur (filtreleme için pay).

Örnekleme ile ses dalgasını kaydedebiliyoruz ancak bu neredeyse sonsuz bir hassasiyet seviyesine doğru kayar, bunun için her örnek belirli bir seviyeye yuvarlanır(quantization). Bu sayede hem insan sesinin ayırt edemeyeceği belirlir sesler tek bir ses olarak verilir hemde depolanması daha kolay olur. Tahmin edebileceğiniz üzere bu örnekleme sayısı ne kadar fazla olursa ses o kadar akıcı ve net duyulabilir.

Örneğin 8 bit derinlikteki bir ses her örnek için 256 farklı seviyesi bulunurken 16 bit derinlik için bu değer yaklaşık 65536'dır. Tabi örnek ve seviyelerin artması kapladığı alanı da arttıracaktır. Tabi bilgisayarlarda ses sadece zaman ve genlik bilgisinden ibaret değildir, ses birden fazla kanalda kaydedilebilir. Mono ses tek bir kanal, stereo sağ ve sol olmak üzere iki kanal içerirken çok kanallı sistemlerde ise (5.1 - 7.1) her kanal ayrı bir ses akışı olarak depolanır. Kanal sayısı arttıkça her kanal için ayrı örnekleme, dolayısıyla ekstra yer kaplayacaktır.

Bilgisayarlarda ham ses verisi genellikle PCM (Pulse Code Modulation) yöntemiyle saklanır. PCM'de ses, belirli bir örneklemme hızı ve bit derinliği kullanılarak ardışık sayısal değerler halinde depolanır, örneğin:

saniyede 44.100 örnek olarak alınan bir sesi 16 bit ve çift kanal kaydetmek istediğimizde bu değer yaklaşık 1.411.200 bit/s(176 KB/s) olur. Bu sebeple ham ses dosyaları geniş yer kaplar

Ham seste kalite yüksektir ancak depolama ve veri aktarımı verimsiz olması ile genelde stüdyo kayıtları ve sesin işlemesi aşamasında kullanılır. Bilgisayarlarda ise bunun yerine daha düzenli olan WAV AIFF gibi formatlar kullanılır. Bu formatlarda seste kalite kaybı olmaz, ham sese yakındır, hala çok yer kaplar ancak düzenleme ve işleme için daha uygundur. Bu formatlar, ses verisinin yanı sıra örneklemme hızı, bit derinliği ve kanal bilgisi gibi meta verileri de içerir. Yine de bu tür formatlar fazla yer kapladığı için ileri bölümlerde değineceğimiz sıkıştırma yöntemleri bu soruna farklı çözüm yolları geliştirmiştir.

## 1.0 Veri sıkıştırma Nedir ve Neden Gereklidir?

Veri sıkıştırma en basit tabiriyle verinin kapladığı alanı gerekli isterler doğrultusunda belirli bir oranda azaltmaktadır. Sıfır ve birlerin belirli bir kurala göre oluşturduğu baytların farklı türde verileri nasıl bilgisayarların anlayabileceğini şekilde depoladığını önceki başlıklar üzerinde uzun uzun anlatmıştık. O başlıklarda da belirttiğimiz gibi her veri türü kendine özel yöntemlerle bilgileri baytlara sığdırıyor, bu sayede bilgisayar bu verileri işleyip bizim istediğimiz halini ekranda bize gösteriyordu, ancak veriler zamanla gelişti, çeşitlendi ve büydü. İşte bu sorun verilerin sıkıştırılması düşüncesindeki ilk adıma bizi götürüyor.

## 1.1

# Veri Sıkıştırmanın Tarihçesi ve Temel Kavramlar

Verilerin sıkıştırılması sanılanın aksine bilgisayarlardan da önceye iletişimim ve bilgi aktarımının ihtiyacının artmasıyla ortaya çıkmıştır. 20 yy. ortalarına doğru telsiz, telefon ve radyo gibi sistemlerde bant genişliği son derece sınırlıydı. Bu durum aynı bilginin daha az veri kullanarak gönderilmesi fikrini oluşturdu. 1948 yılında Claude E. Shannon tarafından ortaya konan Bilgi Teorisi, sıkıştırmanın teorik temellerini oluşturdu. Shannon, bilginin içерdiği belirsizlik miktarını entropi kavramı ile tanımlamış ve herhangi bir veri kaynağının, belirli bir sınırın altına kayıpsız olarak sıkıştırılamayacağını matematiksel olarak göstermiştir. Bu sınır günümüzde Shannon sınırı olarak bilinmektedir. 1950'li ve 60'lı yıllarda bilgisayarların yaygınlaşması ile birlikte depolama alanlarının son derece pahalı ve kısıtlı olması, veri sıkıştırmayı pratik bir zorunluluk haline getirdi. Bu dönemde geliştirilen ilk yöntem verideki tekrar eden desenleri hedef alan RLE(Run Length Encoding), metin ve ilkel görüntülerde başarılı bir performans göstermiştir.

1970 ve 80'li yıllara gelindiğinde ise olasılık tabanlı kodlama yaklaşımıları ön plana çıkmış, sembollerin bulunma sıklığına göre kısa veya uzun bitlerle temsil edilmiştir. Huffman kodlama bu yaklaşımın en popüler örneklerinden biridir. LZ77 ve LZ78 algoritmaları ile bunlardan türetilen LZW yöntemi, özellikle genel amaçlı dosya sıkıştırında yaygın olarak kullanılmıştır. UNIX sistemlerindeki compress aracı ve erken dönem arşiv formatları bu algoritmalar üzerine kurulmuştur.

1990'lı yıllarda kişisel bilgisayarların ve internetin hızla yaygınlaşması, veri miktarının üstel bir biçimde artmasına neden olmuştur. Artık sıkıştırma yalnızca depolama için değil, iletim hızı, bant genişliği kullanımı ve kullanıcı deneyimi açısından son derece önemli hale gelmiştir. Bu dönemde görüntü ve ses verileri için kayıplı(lossy) sıkıştırma yaklaşımı geliştirilmiştir. İnsan gözüne ve kulağının algılayamadığı ya da daha az hassas olduğu bileşenlerin atılması esasına dayanan bu yöntemler sayesinde yüksek sıkıştırma oranları elde edilmiştir. JPEG, MP3 ve MPEG standartları bu yaklaşımın en önemli ürünleri olmuştur.

Günümüze doğru veri sıkıştırma, tek bir algoritmadan ziyade uygulamaya özel çözümler üretmeye yönelmiştir. Kayıpsız ve kayıplı yöntemlerin birlikte kullanıldığı hibrit yaklaşımlar, video akışı, bulut depolama ve gerçek zamanlı iletişim sistemlerinde yaygınlaşmıştır.

H.264, H.265 (HEVC), AV1 gibi modern video kodlama standartları; ZIP, GZIP, ZSTD gibi gelişmiş dosya sıkıştırma formatları bu dönemin ürünleridir. Günümüzde veri sıkıştırma, makine öğrenmesi ve yapay zekâ destekli yöntemlerle daha da gelişmekte ve yoğun veri içeren sistemlerin vazgeçilmez bir parçası olmaya devam etmektedir. Şimdi isterseniz farklı veri türleri için kullanılan sıkıştırma yöntemlerine biraz daha derinden inceleyelim.

## 2.0 Temel Sıkıştırma Algoritmaları

Yöntemlere giriş yapmadan önce şunu belirtmekte fayda var. Sıkıştırma yöntemleri esasen iki tür olarak karşımıza çıkar, kayıplı ve kayıpsız. Kayıpsız sıkıştırma da verinin içeriği tamamen korunur ancak sıkıştırma miktarı azdır, özellikle metinsel ifadelerde eksik veri tüm metnin yada metinsel ifadenin anlamını bozabileceğinden kayıplı sıkıştırma kullanılmaz. Kayıplı sıkıştırma ise insan algılarının zayıf noktalarına odaklanarak verinin bazı bölümlerini tamamen değiştirir. Bu değerlendirme veriyi sıkıştırmayı kolaylaştırır bu sayede kayıpsız sıkıştırma göre daha iyi bir sıkıştırma oranına sahiptir ancak veri artık eski veri değildir. Bu nedenle belirli bir orandaki veri kaybının önemsenmeyeceği veri aktarımı yada depolanması için kullanılır. Bu nedenle görüntü ve sesin sıkıştırılmasında kayıplı sıkıştırma sıklıkla tercih edilir.

## 2.1 Metinlerin sıkıştırılması

Claude Shannon'un bilgi teorisine dayanarak ortaya çıkan **Shannon-Fano** kayıpsız sıkıştırma yöntemi temelde sık geçen sembollerin daha kısa bitlerle, daha seyrek geçen sembollerin ise daha uzun bitlerle temsil edilmesine dayanır.

### SHANNON-FANO COMPRESSION

| Symbol | Frequency | Code | Code |
|--------|-----------|------|------|
| A      | 0.4       |      | 0    |
| B      | 0.4       |      | 10   |
| C      | 0,1       |      | 110  |
| D      | 0,1       |      | 111  |

Semboller olasılıklarına göre sıralanır. Olasılıkları yakın olanlar iki gruba bölünür, her gruba sırayla bit ataması yapılır. Alt gruplar için de aynı işlem tekrarlanır. Bu yöntemdeki en büyük özelliklerden birisi ön ek özelliğine sahip olmasıdır. Hiçbir sembolün kodu, başka bir sembolün kodunun başı değildir. Bu sayede ayırcı karaktere gerek kalmadan kod çözme mümkündür.

Shannon-Fano kodlama, pratikte günümüzde çok yaygın kullanılmasa da, modern entropi tabanlı sıkıştırma algoritmalarının öncüsü olarak kabul edilir. Özellikle Huffman kodlama, bu yöntemin eksiklerini gidererek geliştirilmiştir.

**RLE(Run Length Encoding)**, veri sıkıştırmanın temel mantığını en yalın hâliyle gösteren yöntemlerden biridir. Günümüzde çoğu modern sıkıştırma algoritması, RLE benzeri tekrar yakalama fikirlerini daha gelişmiş biçimlerde kullanmaktadır.

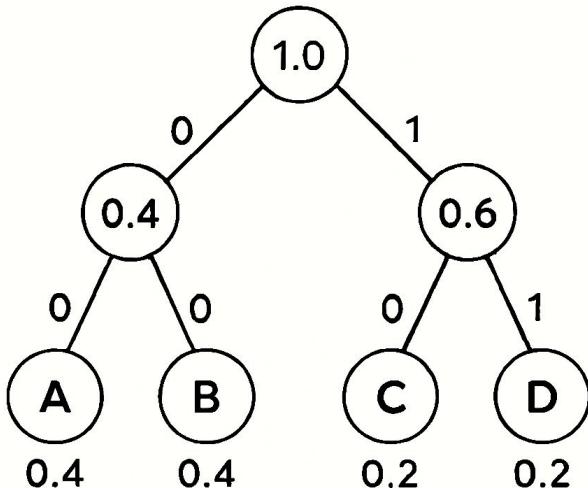
Aslında oldukça basit bir mantığa sahip olan RLE yönteminde aynı sembollerin birden fazla tekrarına, örneğin AAAAVVVVDDSS gibi, karşı tekrar sayısına göre hangi simbol kaç kere tekrar ediyorsa o sayı ve simbol yazılır(3A4V3D2S).

RLE yöntemi uygulaması kolay, hesaplama maliyeti düşük, ve tekrarlı verilerde yüksek sıkıştırma oranı sağlar, ancak tekrarlı olmayan veriler, doğla metinler ve karmaşık görüntüler için verimsizdir.

**Huffman kodlama**, kayıpsız ve olasılık tabanlı bir veri sıkıştırma algoritmasıdır. 1952 yılında David A. Huffman tarafından geliştirilmiş olan bu yöntem, sembollerin veri içerisindeki görülmeye sıklıklarını temel alır. Algoritmanın temel amacı, sık karşılaşılan sembollerı daha kısa bit dizileriyle, daha az görülen sembollerı ise daha uzun bit dizileriyle temsil ederek ortalama kod uzunluğunu en aza indirmektir.

Bu yöntemde öncelikle veri içerisindeki tüm sembollerin frekansları belirlenir ve bu frekanslara göre bir ağaç yapısı oluşturulur. En düşük frekansa sahip semboller ağaçın alt seviyelerinde yer alırken, daha sık geçen semboller köke daha yakın konumlandırılır. Ağaç yapısı tamamlandıktan sonra, kökten yapraklara doğru izlenen yollar bit dizilerine dönüştürülerek her simbol için benzersiz bir Huffman kodu elde edilir. Bu kodlar ön-ek özelliğine sahip olduğundan, herhangi bir simbolün kodu başka bir simbolün kodunun başlangıcı olmaz ve bu sayede kod çözme işlemi ek ayırcılara ihtiyaç duyulmadan gerçekleştirilebilir.

# HUFFMAN CODING



Huffman kodlama, sabit sembol olasılıkları altında ortalama kod uzunluğunu minimize etmesi nedeniyle Shannon–Fano kodlamaya kıyasla daha verimli sonuçlar üretir. Bu özelliği sayesinde metinsel verilerin sıkıştırılmasında uzun yıllar boyunca yaygın olarak kullanılmış ve birçok modern sıkıştırma algoritmasının temel bileşeni haline gelmiştir. Günümüzde ZIP, PNG ve DEFLATE gibi formatlar içerisinde hâlen aktif olarak yer almaktadır.

İstatistiksel sıkıştırmanın bir diğer önemli yöntemi aritmetik kodlamadır. **Aritmetik kodlama**, sembollerin tek tek bit dizileriyle temsil etmek yerine, tüm mesajı 0 ile 1 arasında bir aralık olarak ifade eder. Semboller işlendikçe bu aralık daraltılır ve ortaya çıkan tek bir sayı, tüm veriyi temsil eder. Bu yöntem, teorik olarak entropi sınırına Huffman kodlamaya kıyasla daha yakın sonuçlar verebilmekte, ancak hesaplama karmaşıklığı nedeniyle uzun süre sınırlı kullanım alanına sahip olmuştur.

**Range coding** ise aritmetik kodlamanın pratikte daha verimli ve uygulanabilir bir türevi olarak geliştirilmiştir. Aritmetik kodlamaya benzer şekilde aralık daraltma prensibini kullanır, ancak hesaplamaları daha basit hâle getirerek donanım ve yazılım uygulamalarında avantaj sağlar. Bu nedenle bazı modern sıkıştırma sistemlerinde tercih edilmektedir.

Sözlük tabanlı sıkıştırma yöntemleri, veri içerisinde tekrar eden sembol dizilerinin tespit edilerek bu dizilerin daha kısa gösterimlerle ifade edilmesine dayanır. Bu yaklaşılarda temel fikir, verideki tekrarların açıkça yeniden yazılması yerine, daha önce görülmüş olan dizilere yapılan referansların kullanılmasıdır. Böylece veri boyutu, tekrar sayısı arttıkça önemli ölçüde azaltılabilir. Bu yöntemler kayıpsız sıkıştırma sınıfında yer almaktır, özellikle metinsel veriler ve genel amaçlı dosya sıkıştırma uygulamalarında yaygın olarak kullanılmıştır.

Bu yaklaşımın ilk örnekleri Lempel ve Ziv tarafından geliştirilen LZ77 ve LZ78 algoritmalarıdır. LZ77, sıkıştırma sırasında kaydırmalı bir pencere kullanarak geçmişteki veriyle karşılaştırma yapar ve tekrar eden dizileri mesafe-uzunluk çiftleriyle ifade eder. LZ78 ise veriyi art arda okurken bir sözlük oluşturur ve tekrar eden dizileri bu sözlükteki indeksler aracılığıyla temsil eder. Her iki yöntemde de sıkıştırma işlemi sırasında sözlük dinamik olarak oluşturulur ve kod çözücü tarafından da aynı yapı yeniden inşa edilir.

LZ78 tabanlı olarak geliştirilen LZW algoritması, sözlük yapısını daha verimli hâle getirerek ek karakter göndermeye gerek kalmadan yalnızca sözlük indekslerinin iletilmesini sağlar. Bu sayede hem sıkıştırma verimliliği artmış hem de uygulama kolaylaşmıştır. LZW, özellikle erken dönem dosya sıkıştırma yazılımlarında ve GIF gibi formatlarda uzun yıllar boyunca yaygın olarak kullanılmıştır.

Zamanla sözlük tabanlı yöntemler, istatistiksel kodlama teknikleriyle birleştirilerek daha gelişmiş algoritmalar ortaya çıkmıştır. DEFLATE algoritması, LZ77'nin tekrar yakalama yeteneğini Huffman kodlama ile birleştirerek hem yüksek sıkıştırma oranı hem de hızlı çözme süresi sunmuştur. Benzer şekilde LZMA ve Zstandard gibi modern algoritmalar, daha büyük sözlükler ve gelişmiş eşleme teknikleri kullanarak günümüz veri yoğun uygulamalarında yüksek performans sağlamaktadır.

## 2.2 Görüntünün Sıkıştırılması

Kayıpsız görüntü sıkıştırma yöntemleri, sıkıştırma işlemi sonrasında görüntünün orijinal hâlinin eksiksiz olarak geri elde edilmesini hedefler. Bu nedenle özellikle tıbbi görüntüleme, bilimsel analiz, teknik çizimler ve arşivleme gibi alanlarda, en küçük bir veri kaybının dahi kabul edilemediği durumlarda tercih edilir. Bu yöntemler, görüntüdeki tekrar eden desenleri ve istatistiksel fazlalıkları ortadan kaldırarak dosya boyutunu azaltmayı amaçlar.

Bu alandaki en temel yöntemlerden biri **Run-Length Encoding (RLE)**'dır. RLE, görüntüde ardışık olarak tekrar eden aynı renk veya piksel değerlerini, tekrar sayısı ve piksel değeri şeklinde temsil eder. Özellikle tek renkli alanların yoğun olduğu siyah-beyaz veya düşük renk derinliğine sahip görüntülerde oldukça etkilidir. Bu nedenle erken dönem görüntü formatlarında ve faks sistemlerinde yaygın olarak kullanılmıştır.

Daha gelişmiş kayıpsız görüntü sıkıştırma yöntemlerinden biri PNG formatında kullanılan tekniktir. **PNG**, görüntü satırları üzerinde tahmin (prediction) işlemi uygulayarak pikseller arasındaki farkları küçültür ve bu farkları DEFLATE algoritması ile sıkıştırır. Bu yaklaşım, doğal görüntülerde bile kayıpsız olarak tatmin edici sıkıştırma oranları elde edilmesini sağlar ve PNG'yi web ortamında yaygın bir format hâline getirmiştir.

JPEG standardının kayıpsız sürümü de kayıpsız görüntü sıkıştırma yöntemlerine bir örnek olarak gösterilebilir. **Lossless JPEG**, pikseller arasındaki farkları tahmin tabanlı yöntemlerle kodlayarak veri bütünlüğünü korur. Her ne kadar kayıplı JPEG kadar yaygın olmasa da, özellikle kalite kaybının kabul edilemediği profesyonel uygulamalarda tercih edilmiştir.

Genel olarak kayıpsız görüntü sıkıştırma yöntemleri, görüntü kalitesini tamamen koruma avantajı sunarken, sağladıkları sıkıştırma oranı kayıplı yöntemlere kıyasla daha sınırlıdır.

Kayıplı görüntü sıkıştırma teknikleri, görüntü verisinin insan gözü tarafından algılanması zor veya önemsiz kabul edilen kısımlarının bilinçli olarak elenmesi esasına dayanır. Bu yaklaşımın temel amacı, görüntüde oluşabilecek küçük kalite kayıplarını kabul edilebilir sınırlar içinde tutarak, dosya boyutunu mümkün olan en düşük seviyeye indirmektir. Bu nedenle kayıplı sıkıştırma yöntemleri, özellikle depolama alanı ve iletim bant genişliğinin sınırlı olduğu uygulamalarda büyük avantaj sağlar.

**JPEG**, kayıplı görüntü sıkıştırma tekniklerinin en yaygın ve en bilinen örneklerinden biridir. Özellikle doğal fotoğrafların depolanması ve internet üzerinden iletilmesi amacıyla geliştirilmiştir. JPEG sıkıştırmanın temelinde, insan görme sisteminin bazı ayrıntılara diğerlerine göre daha az duyarlı olduğu gerçeği yer alır. Bu sayede görsel kalite kabul edilebilir düzeyde korunurken dosya boyutunda önemli ölçüde azalma sağlanır.

JPEG algoritması, görüntüyü öncelikle renk bileşenlerine ayırarak işler. Renk bilgisinin parlaklığa göre daha az algılanması nedeniyle, renk bileşenleri üzerinde daha agresif bir sıkıştırma uygulanır. Ardından görüntü, küçük bloklara bölünerek her blok üzerinde ayrık kosinüs dönüşümü (Discrete Cosine Transform – DCT) uygulanır. Bu dönüşüm, görüntü bilgisini uzamsal alandan frekans alanına taşıyarak düşük frekanslı bileşenlerde görüntünün temel yapısını, yüksek frekanslı bileşenlerde ise ince detayları toplar.

Dönüşüm sonrasında elde edilen frekans katsayıları nicemleme işlemine tabi tutulur. Nicemleme aşaması, JPEG sıkıştırmanın kayıplı kısmını oluşturan en kritik adımdır. İnsan gözü tarafından fark edilmesi zor olan yüksek frekanslı bileşenler daha kaba biçimde temsil edilir veya tamamen elenir. Bu işlem, sıkıştırma oranını doğrudan belirler ve görüntü kalitesi ile dosya boyutu arasındaki dengeyi sağlar.

Son aşamada nicemlenmiş katsayılar, genellikle Huffman kodlama gibi istatistiksel yöntemlerle kayıpsız olarak sıkıştırılır. Böylece JPEG, algısal kayıplı ve kayıpsız tekniklerin birlikte kullanıldığı hibrit bir yapı sunar. JPEG, yüksek sıkıştırma oranı, geniş donanım ve yazılım desteği ve kabul edilebilir görsel kalite sunması nedeniyle dijital fotoğrafçılıktan web uygulamalarına kadar pek çok alanda uzun yıllardır standart hâline gelmiştir.

Ses sıkıştırma yöntemleri genel olarak kayıpsız ve kayıplı olmak üzere iki ana gruba ayrılır. Kayıpsız ses sıkıştırma, ses sinyalinin sıkıştırma sonrasında orijinal hâliyle eksiksiz olarak geri elde edilmesini hedefler. Bu nedenle profesyonel ses arşivleme, stüdyo kayıtları ve kalite kaybının kabul edilemediği uygulamalarda tercih edilir. FLAC, ALAC ve WavPack gibi formatlar, ses verisindeki istatistiksel fazlalıkları ortadan kaldırarak dosya boyutunu azaltır; ancak sağladıkları sıkıştırma oranı kayıplı yöntemlere kıyasla daha sınırlıdır.

Kayıplı ses sıkıştırma yöntemleri ise insan işitme sisteminin algısal özelliklerini temel alır. İnsan kulağının belirli frekans aralıklarına daha duyarlı olması ve bazı seslerin diğerleri tarafından maskelenmesi, bu yöntemlerin temelini oluşturur. Bu sayede iştilmesi zor veya önemsiz kabul edilen ses bileşenleri bilinçli olarak atılarak yüksek sıkıştırma oranları elde edilir.

Bu alandaki en yaygın ve tarihsel açıdan en önemli yöntem MP3 (MPEG-1 Audio Layer III)'tür. MP3, ses sinyalini frekans bileşenlerine ayırarak psikoakustik modeller kullanır ve insan kulağının algılayamayacağı ya da maskelenmiş frekansları ortadan kaldırır. Sinyal daha sonra zaman-frekans alanında nicemlenir ve verimli biçimde kodlanır. MP3, sunduğu kabul edilebilir ses kalitesi, yüksek sıkıştırma oranı ve geniş donanım-yazılım desteği sayesinde 1990'lı yillardan itibaren dijital müziğin yaygınlaşmasında kritik bir rol oynamıştır.

MP3'ün ardından geliştirilen AAC, Ogg Vorbis ve Opus gibi formatlar, benzer algısal prensipleri daha gelişmiş kodlama teknikleriyle uygulayarak daha yüksek verimlilik sağlamayı hedeflemiştir. Bu formatlar, genellikle

MP3'e kıyasla daha düşük bit hızlarında benzer ya da daha iyi ses kalitesi sunamalarına rağmen, MP3'ün yaygınlığı ve tarihsel önemi hâlâ belirleyici konumdadır.

Sonuç olarak, ses sıkıştırma alanında kayıpsız yöntemler kaliteyi koruma amacıyla, kayıplı yöntemler ise veri boyutu ve iletim verimliliğini ön planda tutacak şekilde geliştirilmiştir. Özellikle MP3, kayıplı ses sıkıştırmanın en bilinen ve en etkili örneklerinden biri olarak bu alanda önemli bir dönüm noktası oluşturmuştur.

### 3. Kapanış

Görüldüğü üzere, günlük hayatımda karşılaştığımız fiziksel verilerin dijital ortama aktarılması süreci tek bir adımdan ibaret değildir; aksine bu süreç, uzun yıllar boyunca pek çok araştırmacının katkılarıyla şekillenmiş çok katmanlı bir yapıya sahiptir. Günümüzde aktif olarak kullandığımız internet, sosyal medya platformları, yapay zekâ uygulamaları, web servisleri, metinler, sesler ve görüntüler; arka planda karmaşık dönüşüm, temsil ve sıkıştırma aşamalarından geçerek bize sunulmaktadır. Ancak çoğu zaman bu verilerin nasıl üretildiği, nasıl saklandığı ve bir bilgisayar sistemiyle nasıl anlamlı bir iletişim kurulduğu göz ardı edilmektedir. Hızla gelişen ve dönüşen çağımızda, dijital teknolojileri yalnızca tüketen bir konumda olmak yerine, bu sistemlerin arkasındaki temel prensipleri anlayabilmek giderek daha önemli hâle gelmektedir. Bu sunumun, dijital dünyanın arka planında işleyen mekanizmlara dair küçük de olsa bir farkındalık oluşturmasını ve eleştirel bir bakış kazandırmamasını umuyorum. Dinledığınız için teşekkür ederim.