Bursa Teknik Üniversitesi
Bilgisayar Mühendisliği
BLM101 – Bilgisayar Mühendisliğine Giriş
Veri Depolama ve Sıkıştırma Algoritmaları
Run-Length Encoding (RLE)
Ayşe Nur Baştuğ 24360859041

İÇİNDEKİLER

VERİ VE BİLGİ

- **Veri**, bilgisayarda işlenebilen ham bilgilerdir.
- Bilgisayarlar tüm verileri **dijital ortamda** saklar.
- Dijital ortamda veriler **0 ve 1'lerden oluşan ikili (binary) sistem** ile temsil edilir.
- **0 ve 1**, bilgisayarın donanımındaki **açık / kapalı** durumları ifade eder.
- Tüm metin, resim ve ses verileri bu **bit dizilerine** dönüştürülerek saklanır.

BİT (BIT) KAVRAMI

- **Bit**, bilgisayarda bilgiyi temsil eden en küçük veri birimidir.
- Yalnızca **0 ve 1** değerlerini alabilir ve ikili sistemi oluşturur.
- Bilgisayarın tüm işlemleri bu iki değer üzerinden gerçekleştirilir.
- Metin, resim, ses ve video gibi tüm dijital veriler **bit dizileri** halinde saklanır.

BYTE KAVRAMI

- **1 byte**, 8 bitten oluşan bir veri birimidir.
- Byte'lar, bitlere göre daha anlamlı veri grupları oluşturur.
- **Harfler, sayılar ve semboller** bilgisayarda byte'lar ile temsil edilir.
- Birçok karakter kodlama sisteminde **1 karakter yaklaşık 1 byte** yer kaplar.
- Metin dosyalarının boyutu, kullanılan **byte sayısına** göre belirlenir.

METİN VERİSİNİN TEMSİLİ

- **Metin verileri**, bilgisayarda **sayısal kodlar** ile temsil edilir.
- Bu kodlama sistemlerinden en yaygın olanlardan biri **ASCII**'dir.
- ASCII'de her **harf, rakam ve sembolün** kendine ait bir sayısal değeri vardır.
- Bilgisayarlar metinleri bu **sayısal karşılıklar** üzerinden işler ve saklar.

ASCII ÖRNEĞİ

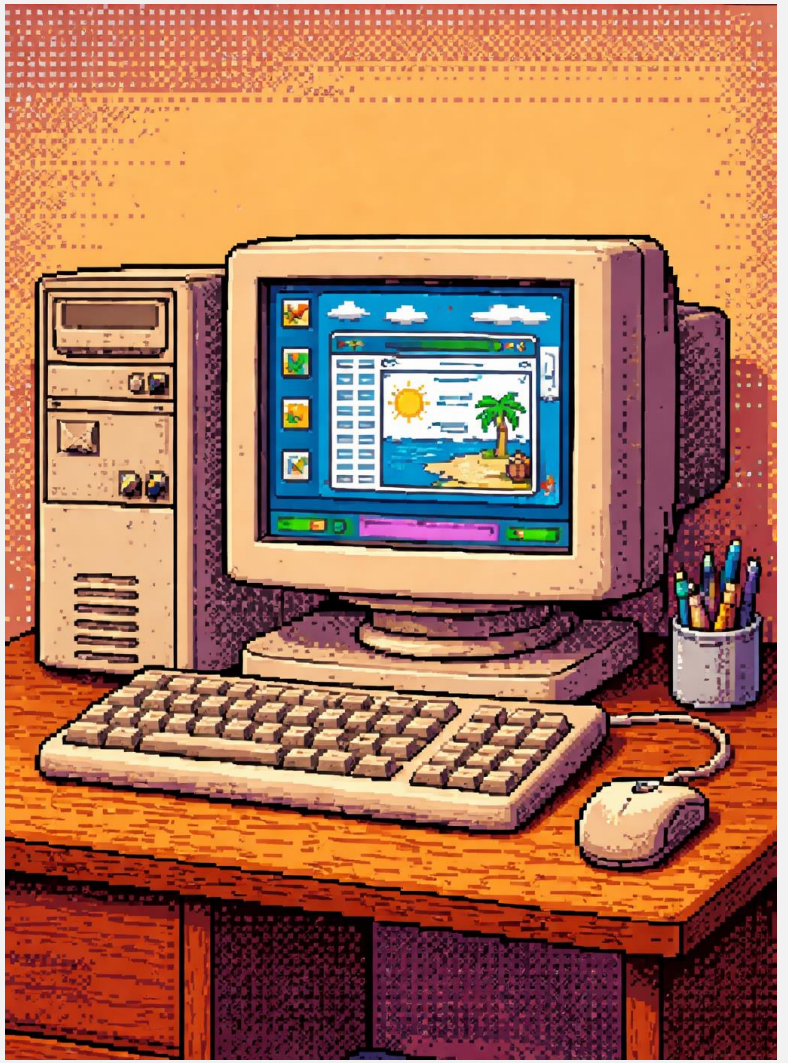
ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- **'A' karakterinin ASCII değeri 65**'tir.
- Bu değer ikili (binary) sistemde 01000001 şeklinde gösterilir.
- Bilgisayarlar karakterleri **sayısal ve ikilik karşılıklarıyla** saklar.
- Tüm metinler, bu tür **binary kodların birleşimiyle** oluşturulur.

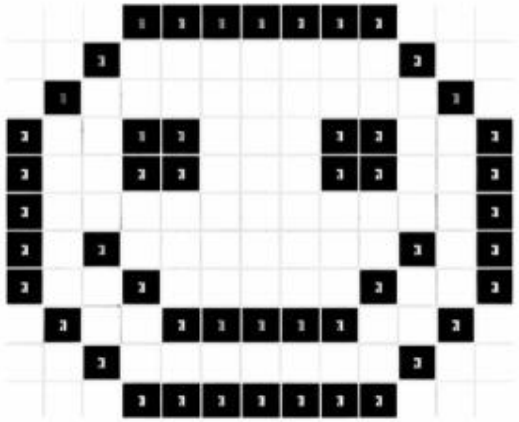
RESİM VERİSİNİN TEMSİLİ

- **Resimler**, çok sayıda **pikselin** bir araya gelmesiyle oluşur.
- Her piksel, kendine ait bir **renk bilgisi** taşır.
- Piksel renkleri **sayısal değerlerle** ifade edilir.
- Bu sayısal değerler, bilgisayar tarafından **0 ve 1'lere** dönüştürülerek saklanır.



SİYAH-BEYAZ GÖRSELLER

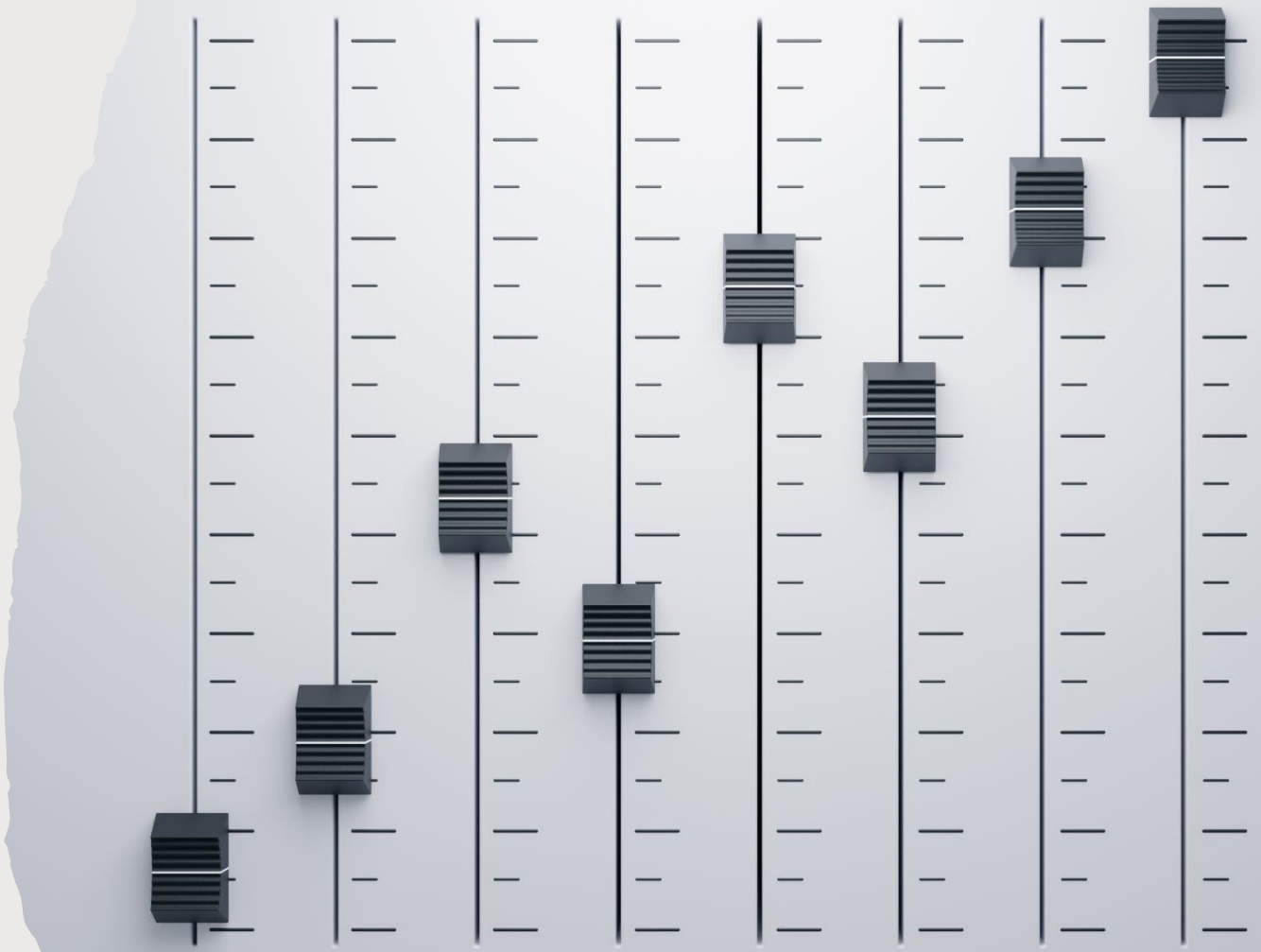
- **Siyah-beyaz resimler**, yalnızca **0** ve **1** kullanılarak temsil edilebilir.
- **0** değeri **siyah**, **1** değeri **beyaz** rengi ifade eder.
- Her piksel bir bit ile gösterilir.
- Görüntü, piksellerden oluşan bir **matris yapısı** şeklinde saklanır.



0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	0
0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

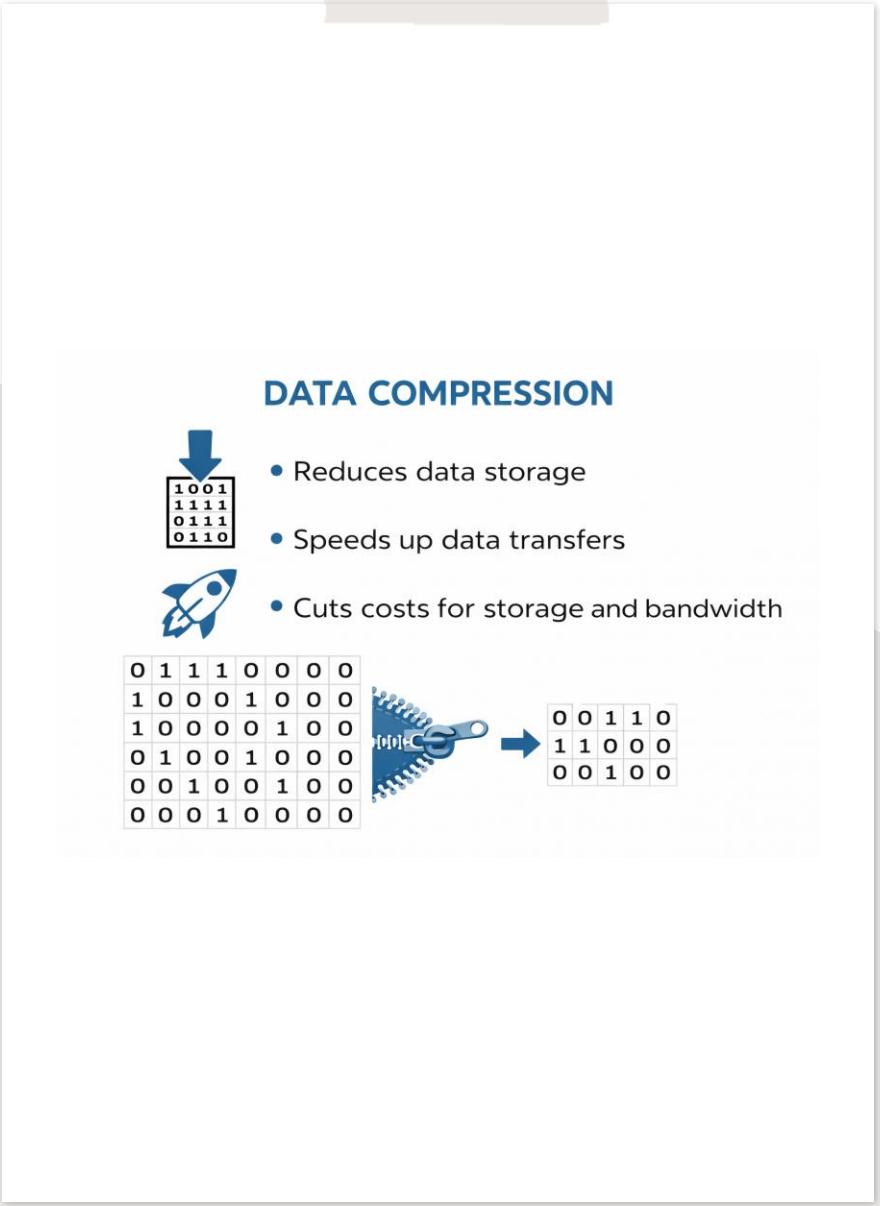
SES VERİSİNİN TEMSİLİ

- **Ses**, doğası gereği **analog bir sinyaldir**.
- Bilgisayarda işlenebilmesi için **dijital hâle getirilir**.
- Bu dönüşüm sırasında **örnekleme (sampling)** yapılır.
- Ardından ses sinyali **sayısallaştırılarak** 0 ve 1'lere çevrilir.



VERİ NEDEN SIKIŞTIRILIR?

- **Depolama alanını azaltmak** için veri sıkıştırma kullanılır.
- Daha az veri sayesinde **veri aktarımı hızlanır**.
- Daha az depolama ve bant genişliği kullanımı ile **maliyetler düşürülür**.



VERİ SIKIŞTIRMA TÜRLERİ

- **Kayıplı sıkıştırma**, verinin bir kısmını kalıcı olarak siler.
- **Kayıpsız sıkıştırma**, verinin tamamını korur.
- Hangi yöntemin kullanılacağı **kullanım alanına göre** belirlenir.

KAYIPSIZ SIKIŞTIRMA

- **Sıkıştırma sonrası orijinal veri eksiksiz olarak geri elde edilir.**
- Veri kaybı kabul edilemediği için **metin ve program dosyalarında** kullanılır.
- **RLE (Run-Length Encoding), kayıpsız veri sıkıştırma** yöntemlerine örnektir.

VERİ TEKRARI PROBLEMİ

- Bazı veri türlerinde **aynı değerler art arda tekrar eder**.
- Bu tekrarlar, verinin boyutunu gereksiz yere **artırır**.
- Tekrar eden veriler aslında **bilgi açısından fazlalık** oluşturur.
- **Sıkıştırma algoritmaları**, bu fazlalıkları tespit ederek veriyi daha **küçük boyutta** saklamayı amaçlar.

RUN-LENGTH ENCODING (RLE)

- **Basit bir veri sıkıştırma algoritmasıdır.**
- **Ardışık tekrar eden** verileri tespit eder ve sayar.
- Tekrar eden verileri **sayı + veri** biçiminde ifade eder.
- Bu sayede veri boyutu **azaltılmış** olur.

RLE NASIL ÇALIŞIR?

Aynı karakterler ardışık olarak **gruplanır**.

Her grubun **tekrar sayısı** hesaplanır.

Elde edilen bilgiler **yeni bir dizi** şeklinde yazılır.

Bu işlem, verinin **daha az yer kaplamasını** sağlar.

RLE ÖRNEĞİ

Girdi: AAAAABBBCCDAA

Çıktı: 5A3B2C1D2A

Orijinal satır:

000011110000

RLE sonucu:

4×0, 4×1, 4×0

RLE AVANTAJLARI

- **Uygulaması kolay** bir algoritmadır.
- Basit işlemler içerdiği için **hızlı çalışır**.
- **Basit veri yapıları** kullanılarak rahatlıkla uygulanabilir.
- Küçük ve tekrar içeren verilerde **etkilidir**.

RLE DEZAVANTAJLARI

Her veri türü için uygun değildir.

Ardışık tekrar içermeyen verilerde sıkıştırma oranı düşüktür.

Örneğin ABCD1234 gibi farklı karakterlerden oluşan bir veri, RLE ile **daha fazla yer kaplayabilir**.

Bu nedenle RLE, yalnızca **tekrar oranı yüksek** verilerde verimli sonuç verir.

PYTHON İLE RLE

- **Python string yapıları**, RLE uygulaması için uygundur.
- Algoritma **döngüler ve fonksiyonlar** kullanılarak yazılır.
- Python ile **okunabilir ve sade** kodlar oluşturulabilir.

```python
def rle(s):
    sonuc = ""
    sayac = 1

    for i in range(1, len(s)):
        if s[i] == s[i-1]:
            sayac += 1
        else:
            sonuc += str(sayac) + s[i-1]
            sayac = 1

    sonuc += str(sayac) + s[-1]
    return sonuc

print(rle("AAABBBCC"))  # 3A3B2C
```

ENCODE FONKSİYONU

- Program, **kullanıcının girdiği veriyi** (metin veya karakter dizisi) giriş olarak alır.
- Girilen verideki **ardışık tekrar eden karakterler** tek tek incelenir.
- Aynı karakterlerin **kaç kez tekrar ettiği** hesaplanır.
- Her tekrar grubu, **tekrar sayısı + karakter** formatında ifade edilir.
- Elde edilen bu bilgiler birleştirilerek **sıkıştırılmış veri çıktısı** oluşturulur.

DECODE FONKSİYONU

- Program, **sıkıştırılmış veriyi** giriş olarak alır.
- Veri içerisindeki **sayı + karakter** yapıları tek tek analiz edilir.
- Her karakter, belirtilen sayı kadar **tekrar edilerek** yeniden oluşturulur.
- Bu işlem sonucunda **orijinal veri eksiksiz şekilde** geri elde edilir.
- Böylece **kayıpsız bir dönüşüm** sağlanmış olur.

SIKIŞTIRMA ORANI

- **Sıkıştırma oranı**, uygulanan sıkıştırmanın ne kadar başarılı olduğunu gösterir.
- Bunun için **orijinal veri boyutu** ile **sıkıştırılmış veri boyutu** karşılaştırılır.
- Boyutlar arasındaki fark, sıkıştırmanın **etkinliğini** ifade eder.
- Sıkıştırma oranı genellikle **yüzde (%)** cinsinden hesaplanır.

PYTHON KOD YAPISI

- **`encode()` fonksiyonu**, orijinal veriyi alarak **sıkıştırma işlemini** gerçekleştirir.
- Bu fonksiyon, ardışık tekrar eden karakterleri sayar ve veriyi **sıkıştırılmış formata** dönüştürür.
- **`decode()` fonksiyonu**, sıkıştırılmış veriyi alır ve **orijinal veriyi geri oluşturur**.
- Böylece verinin kayıpsız olarak çözümlenmesi sağlanır.
- **Ana program bloğu**, kullanıcıdan veri alır, `encode()` ve `decode()` fonksiyonlarını çağırır ve **sonuçları**

```python
def encode(s):
    res, count = "", 1
    for i in range(1, len(s)):
        if s[i]==s[i-1]: count+=1
        else: res+=str(count)+s[i-1]; count=1
    res+=str(count)+s[-1]
    return res

def decode(s):
    return "".join(int(s[i])*s[i+1] for i in range(0,len(s),2))

# Ana program
girdi = "AAABBC"
print("Encode:", encode(girdi))
print("Decode:", decode(encode(girdi)))
```

UYGULAMA ÇIKTISI

- Program, **kullanıcının girdiği veriyi** alır.
- `encode()` fonksiyonu ile veri **sıkıştırılır** ve sıkıştırılmış hâli elde edilir.
- `decode()` fonksiyonu ile sıkıştırılmış veri **orijinal hâline geri çevrilir**.
- Böylece kullanıcı, **girdi → sıkıştırılmış → geri çözümlenmiş** veriyi görebilir.

ÖRNEK ÇALIŞMA

Girdi: AAAAABBBCCDAA

Encode sonucu: 5A3B2C1D2A

Decode sonucu: AAAAABBBCCDAA

ELDE EDİLEN SONUÇLAR

- **Veri boyutu**, RLE algoritması sayesinde **azaltılmıştır**.
- Sıkıştırma sırasında **orijinal veri eksiksiz olarak korunmuştur**.
- `encode()` ve `decode()` işlemleri, algoritmanın **doğru çalıştığını** göstermektedir.
- Bu sayede hem **depolama alanı tasarrufu** sağlanmış hem de **kayıpsız veri dönüşümü** gerçekleştirilmiştir.

DEĞERLENDİRME

- **RLE**, öğrenmesi ve uygulaması kolay, **basit bir sıkıştırma algoritmasıdır**.
- Ardışık tekrarları sayarak veri boyutunu azaltır ve **temel sıkıştırma mantığını** gösterir.
- Bu algoritma, **daha karmaşık veri sıkıştırma yöntemlerinin** anlaşılmasına da yardımcı olur.
- Küçük ve tekrar içeren verilerde **etkili ve öğretici bir örnek** olarak kullanılır.

GELİŞTİRİLEBİLİR YÖNLER

- **Dosya üzerinden sıkıştırma:** RLE ve diğer sıkıştırma algoritmaları, **metin veya veri dosyalarına** uygulanabilir.
- **Görsel veriye uygulama:** Piksel tekrarları içeren siyah-beyaz veya basit renkli görsellerde **RLE etkili olabilir**.
- **Farklı algoritmalarla karşılaştırma:** RLE, **kayıplı ve kayıpsız diğer yöntemlerle** performans ve verimlilik açısından kıyaslanabilir.
- Böylece algoritmanın **avantajları, sınırlamaları ve kullanım alanları** daha iyi anlaşılır.

KAYNAKÇA

- Text and Image Compression based on Data Mining Perspective
- Run-Length Encoding (RLE) – Wikipedia
- Python Resmi Dokümantasyonu
- İlgili Dersin Kitabı (Bilgisayar Bilimine Giriş)-Chapter 1