Veri Depolama ve Sıkıştırma Algoritmaları

Sunum İçeriği

- **Bölüm 1:** Veri Temsili (Bit & Byte)
- **Bölüm 2:** Metin Verileri (ASCII / Unicode)
- **Bölüm 3:** Resim ve Ses Temsili
- **Bölüm 4:** Veri Sıkıştırma İhtiyacı
- **Bölüm 5:** RLE Algoritması (Teori)
- **Bölüm 6:** Python Projesi (Uygulama)

Verinin Doğası: Analog vs Dijital

Analog

Doğadaki veriler süreklidir (Continuous). Ses dalgaları, ışık spektrumu, sıcaklık değişimi kesintisizdir.

Dijital

Bilgisayar verileri ayrıktır (Discrete). Sürekli veriler belirli aralıklarla örneklenerek sayılara dönüştürülür.

En Küçük Yapı Taşları

Bit (Binary Digit)

Bilgisayarın anlayabildiği en küçük birimdir. Elektrik var (1) veya yok (0) durumunu temsil eder.

1 Bit

Byte

8 adet bitin bir araya gelmesiyle oluşur. Bellek adreslemesinin temel birimidir.

```
10110010
```

1 Byte (8 Bit)

Neden 2'lik Sistem?

Bilgisayarlar elektronik devrelerden oluşur. Transistörlerin iki kararlı durumu vardır: Açık veya Kapalı.

Kombinasyon Hesabı:

N adet bit ile 2^N farklı durum ifade edilebilir.

---

1 Bit ---> 2^1 = 2 Durum (0, 1)

8 Bit ---> 2^8 = 256 Durum (0 - 255)

1. Metin Verisinin Temsili

Sayısal Eşleştirme (Mapping)

Bilgisayar "A" harfini bilmez. "A" harfine karşılık gelen sayıyı bilir.

Tarih boyunca farklı harita yöntemleri (Encoding) geliştirilmiştir:

- ASCII
- Extended ASCII
- Unicode (UTF-8)

ASCII Standardı

American **S**tandard **C**ode for **I**nformation **I**nterchange

Özellikleri

- 1960'larda geliştirildi.
- 7 Bit kullanır (2^7 = 128 karakter).
- İngilizce alfabesi, rakamlar ve noktalama işaretlerini kapsar.

```
'A' → 65 (01000001) 'B' → 66 (01000010)
'a' → 97 (01100001) ' ' → 32 (00100000)
```

Unicode ve UTF-8

Evrensel İhtiyaç

ASCII, Türkçe (ğ, ş), Çince veya Arapça karakterleri desteklemez. Bu sorunu çözmek için **Unicode** geliştirildi.

UTF-8 (8-bit Unicode Transformation Format)

Değişken uzunluklu kodlama kullanır:

- Standart İngilizce karakterler: **1 Byte**
- Türkçe karakterler (ğ, ü, ş): **2 Byte**
- Emojiler (😊): **4 Byte**

2. Görüntü Verisinin Temsili

Piksel (Picture Element)

Dijital bir görüntü, ızgara (matris) yapısındaki renkli noktalardan oluşur.

Bir görüntünün kalitesini ve boyutunu iki faktör belirler:

1. **Çözünürlük:** Yatay x Dikey piksel sayısı.
2. **Renk Derinliği:** Her pikselin rengini saklamak için kullanılan bit sayısı.

Renk Derinliği: Siyah-Beyaz

1-Bit Görüntü (Monokrom)

0 1

Her piksel ya 0 (Siyah) ya da 1 (Beyaz) olabilir.

8-Bit Gri Tonlama (Grayscale)

Her piksel 0-255 arasında bir değer alır. (0=Siyah, 255=Beyaz)

Renk Derinliği: RGB (Renkli)

Bilgisayarlar renkleri 3 ana rengin karışımı olarak saklar: **Kırmızı (R), Yeşil (G), Mavi (B)**.

True Color (24-bit)

- R Kanalı: 8 Bit (256 Ton)
- G Kanalı: 8 Bit (256 Ton)
- B Kanalı: 8 Bit (256 Ton)

Toplam: 256 x 256 x 256 = **Y**aklaşık 16.7$ Milyon Renk

Örnek: Görüntü Boyutu Hesaplama

Soru: 1920x1080 çözünürlüğünde, 24-bit renk derinliğine sahip bir fotoğrafın ham boyutu nedir?

1. Piksel Sayısı:

1920 x 1080 = 2,073,600 piksel

2. Toplam Bit:

2,073,600 x 24 bit = 49,766,400 bit

3. Byte Dönüşümü:

49,766,400 / 8 = 6,220,800 Byte

4. Sonuç:

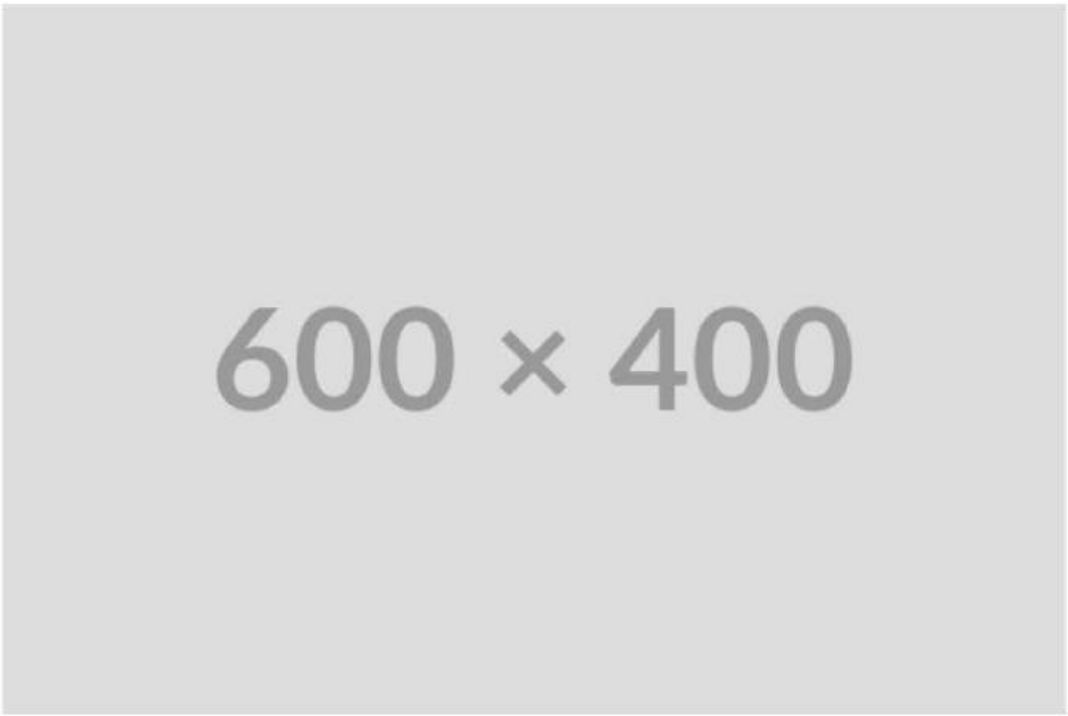
Yaklaşık 6.22 MB (Megabyte)

3. Ses Verisinin Temsili

Analogdan Dijitale

Ses, havada yayılan bir dalgadır (Analog). Bilgisayar bunu saklamak için belirli aralıklarla sesin "fotoğrafını çeker".

Bu işleme **Örnekleme (Sampling)** denir.



Kriter 1: Örnekleme Hızı (Hz)

Saniyede kaç kez örnek alındığını belirtir.

8,000 Hz

Telefon Görüşmesi

(Düşük Kalite)

44,100 Hz

CD Kalitesi

(Standart)

96,000 Hz

Stüdyo Kaydı

(Yüksek Kalite)

Kriter 2: Bit Derinliği (Quantization)

Her bir ses örneğinin ses şiddetini (amplitude) kaydetmek için kullanılan bit sayısıdır.

- **8-bit Ses:** 256 ses seviyesi. (Cızırtılı, eski oyun sesleri)
- **16-bit Ses:** 65,536 ses seviyesi. (CD standartı, net ses)
- **24-bit Ses:** 16 milyon ses seviyesi. (Profesyonel kayıt)

Formül: Hız x Bit x Kanal(Sayısı) = Saniyelik(Veri)

4. Neden Sıkıştırma?

Ham veriler çok büyük yer kaplar. İnternet hızı ve disk kapasitesi sınırlıdır.

Ham Veri

1 Dakika HD Video: ~10 GB

1000 Fotoğraf: ~30 GB

Sıkıştırılmış Veri

1 Dakika HD Video: ~100 MB

1000 Fotoğraf: ~3 GB

Yöntem A: Kayıpsız (Lossless)

Dosya sıkıştırılıp tekrar açıldığında, **bit-bit orijinalin aynısıdır.** Hiçbir veri kaybolmaz.

Nerelerde Kullanılır?

- Metin Belgeleri (.txt, .docx)
- Veritabanları
- Program Kodları (.py, .exe)
- Tıbbi Görüntüler (Detay kaybı riskli)

Algoritmalar

ZIP

RLE

Huffman

PNG

Yöntem B: Kayıplı (Lossy)

Gereksiz veya insan algısının fark edemeyeceği veriler **kalıcı olarak silinir**. Çok yüksek sıkıştırma sağlar.

Örnekler

- **JPEG:** Gözün ayırt edemediği renk tonlarını atar.
- **MP3:** İnsan kulağının duymadığı frekansları siler.

5. Run-Length Encoding (RLE)

Nedir?

En temel **kayıpsız** sıkıştırma algoritmasıdır.

Mantık: Arka arkaya tekrar eden verileri (Run), verinin kendisi ve tekrar sayısı (Length) olarak saklar.

En İyi Sonuç: Basit grafikler, ikonlar, çizimler.

En Kötü Sonuç: Fotoğraflar, rastgele metinler.

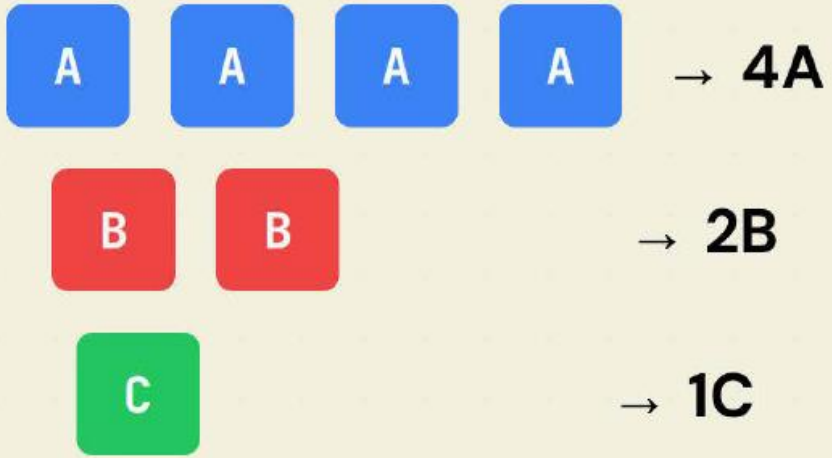
RLE Nasıl Çalışır? - Adım 1

Elimizde sıkıştırılacak şöyle bir veri olsun:

Toplam Boyut: **7 Karakter**

RLE Nasıl Çalışır? - Adım 2

Algoritma grupları sayar:



RLE Nasıl Çalışır? - Sonuç

Girdi (Input)

AAAABBC

Boyut: 7 Birim

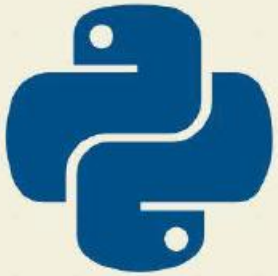
Çıktı (Output)

4A2B1C

Boyut: 6 Birim

Bu basit örnekte 1 karakterlik kazanç sağlandı.

6. Python Projesi



Proje Amacı

Kullanıcıdan bir metin girdisi alan ve bunu RLE algoritması kullanarak sıkıştıran (encode) ve tekrar çözen (decode) bir program yazmak.

Hedef: Algoritmik düşünme ve string manipülasyonu becerilerini geliştirmek.

Algoritma Mantığı (Encode)

1. Boş bir sonuç stringi ("") oluştur.
2. İlk karakteri seç ve sayacı 1 yap.
3. Metni döngüyle gez:
    - Eğer sıradaki karakter öncekine eşitse: **Sayacı artır.**
    - Eğer farklıysa: **Sayacı ve karakteri sonuca ekle. Sayacı 1 yap.**
4. Döngü bitince son grubu da ekle.

Kaynak Kod: Sıkıştırma

```python
def rle_encode(data):
    #hata almamak için güvenlik kontrolü
    if not data:
        return ""

    encode = ""
    bakılan_harf = data[0]
    sayac = 1

    #sıkıştırma döngüsü
    for i in range(1, len(data)):
        if data[i] == bakılan_harf:
            sayac += 1
        else:
            encode += str(sayac) + bakılan_harf
            bakılan_harf = data[i]
            sayac = 1

    # Son grup için döngüden çıktıktan sonra encode metinine ekleme
    encode += str(sayac) + bakılan_harf
    #Fonksiyon sonucunu döndürür.
    return encode
```

Kaynak Kod: Çözme

```
24  def rle_decode(data):
25      #hata almamak için güvenlik kontrolü
26      if not data:
27          return""
28      decode = ""
29      sayac = ""
30      #verideki her karakteri tek tek inceleme
31      for char in data:
32          #bir stringin içeriğinin tamamen rakamlardan oluşup
33          #oluşmadığını kontrol etme (rakam:true  diğer:false)
34          if char.isdigit():
35              sayac += char
36          else:
37              decode += char * int(sayac)
38              sayac = ""
39      return decode
```