

Veri Depolama ve Sıkıştırma Algoritmaları: RLE

Ders: BLM101 - Bilgisayar Mühendisliğine Giriş

Hazırlayan: Görkem GÜNEY

Öğrenci Numarası: 25360859016

Sunum İçeriği

1

Veri Temsili Nedir?

(Bit & Byte)

2

Metin, Resim ve Ses Verileri

Nasıl Saklanır?

3

Veri Sıkıştırma İhtiyacı

4

Sıkıştırma Türleri

(Kayıplı vs Kayıpsız)

5

RLE Algoritması

(Run-Length Encoding)

6

Python Proje Uygulaması

Bilgisayarın Dili: İkili Sistem (Binary)

Bilgisayarlar bizim gibi karmaşık harfleri veya ondalık sayıları doğrudan anlayamazlar. Onların evreni, sadece iki temel durumdan ibarettir: elektrik var (1) ya da elektrik yok (0).

Bu en küçük bilgi birimine **Bit** (Binary Digit) denir.

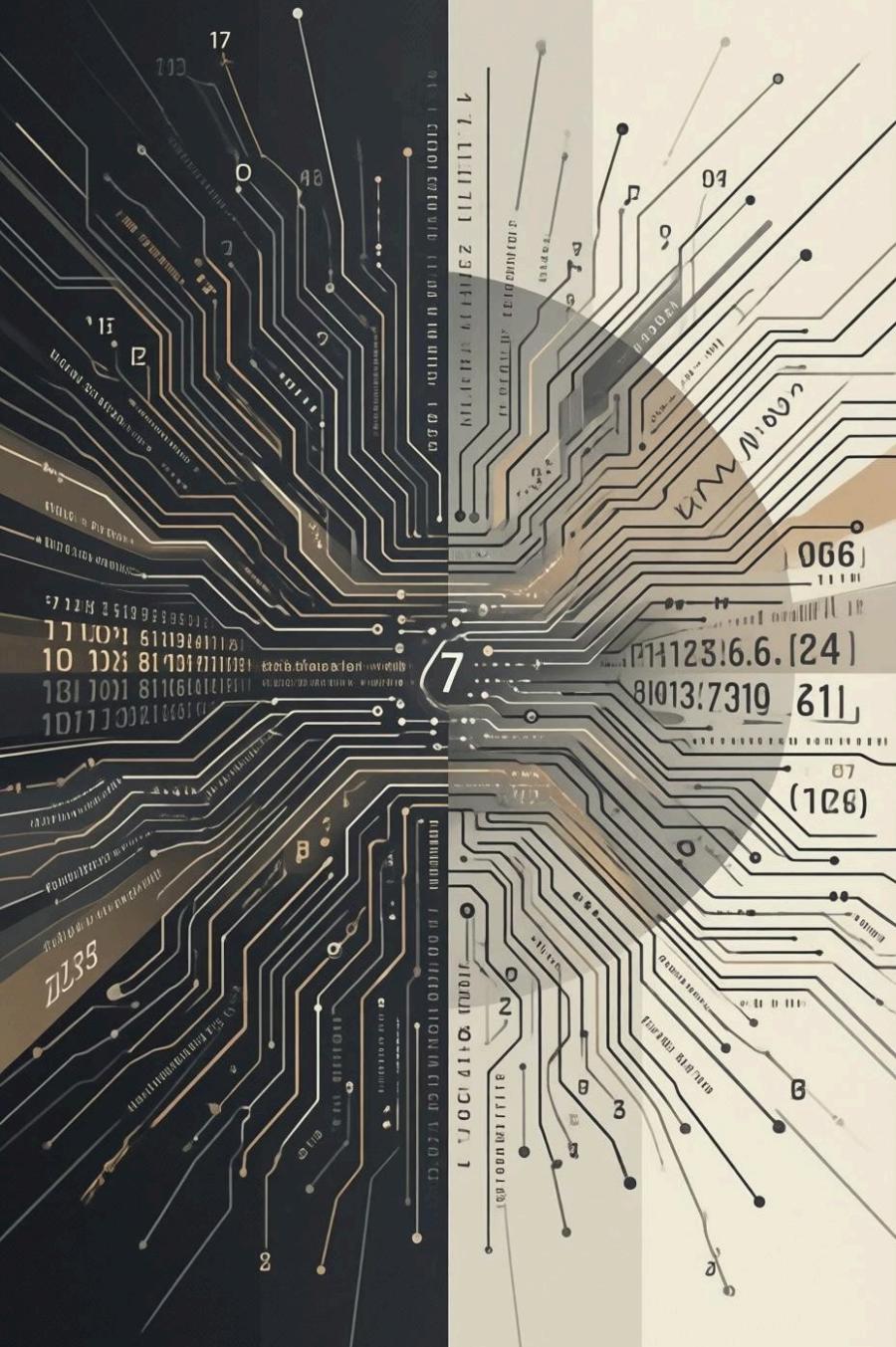
Bit ve Byte Kavramları

Tek bir bit, sadece iki durumu (0 veya 1) temsil edebilir ve bu çok az bilgi taşır. Anlamlı veriler oluşturmak, komutları işlemek ve bilgileri saklamak için bitlerin gruplandırılması gereklidir.

8 Bit = 1 Byte

Bir byte, bilgisayarların çoğu işleminde temel adreslenebilir birimidir. 1 byte ile 256 farklı karakter veya sayı temsil edilebilir (28 farklı kombinasyon).





Veri Nasıl Temsil Edilir?

Bilgisayarda gördüğümüz her şey; izlediğimiz filmler, dinlediğimiz müzikler, okuduğumuz yazılar, aslında arka planda sayılardan ibarettir.

Bu bölümde, metin, resim ve ses gibi farklı veri tiplerinin bilgisayarlar tarafından nasıl sayısallaştırıldığını ve saklandığını derinlemesine inceleyeceğiz.

Metin Verisinin Temsili: ASCII

Klavyede bastığımız her harf, rakam veya sembolün bilgisayar içinde sayısal bir karşılığı vardır. Bu sayısal eşleştirmenin, metin verilerinin dijital ortamda işlenmesini ve depolanmasını sağlar.

- Örneğin, büyük 'A' harfi bilgisayarda **65** sayısıyla temsil edilir.
- 65 sayısının ikili (binary) karşılığı ise: **01000001**'dir.

Bu standart, **ASCII** (American Standard Code for Information Interchange) olarak bilinir ve uzun yıllar metin kodlamasında temel olmuştur.



Unicode: Evrensel Karakter Seti



Küresel İhtiyaçlar

ASCII, sadece İngilizce karakterleri ve 128 farklı karakteri kapsayarak Latin alfabetesini temel alan diller için yeterliydi. Ancak dünya dilleri çok daha çeşitliydi.



Çok Dillilik Sorunu

Türkçe'deki özel karakterler (ç, ğ, ı, ö, ş, ü), Çince'deki binlerce karakter, Arapça ve diğer dillerdeki semboller ASCII ile temsil edilemiyordu.



Emoji Entegrasyonu

Günümüz dijital iletişimiminin vazgeçilmezi olan emojiler de ASCII standardının ötesinde bir kodlama sistemi gerektiriyordu.



Unicode Çözümü

Tüm bu ihtiyaçlara yanıt olarak **Unicode** geliştirildi. Unicode, binlerce farklı karakteri ve simbolü tek bir evrensel standart altında temsil edebilir, böylece dil ve karakter sınırlamalarını ortadan kaldırır.

Resim Verisinin Temsili

Piksellerin Gücü

Dijital resimler, gözle görülemeyecek kadar küçük kare veya daire şeklindeki noktalardan oluşur. Bu noktalara **Piksel** denir. Her bir piksel, resmin belirli bir noktasındaki rengi ve yoğunluğu temsil eder.

Renk Kodları ve Matrisler

Bilgisayar bir resmi aslında "görmez". Onun için bir resim, her bir pikselin renk değerini içeren devasa bir sayı matrisinden (tablosundan) ibarettir. Bu sayılar, resmin tüm görsel bilgilerini kodlar.



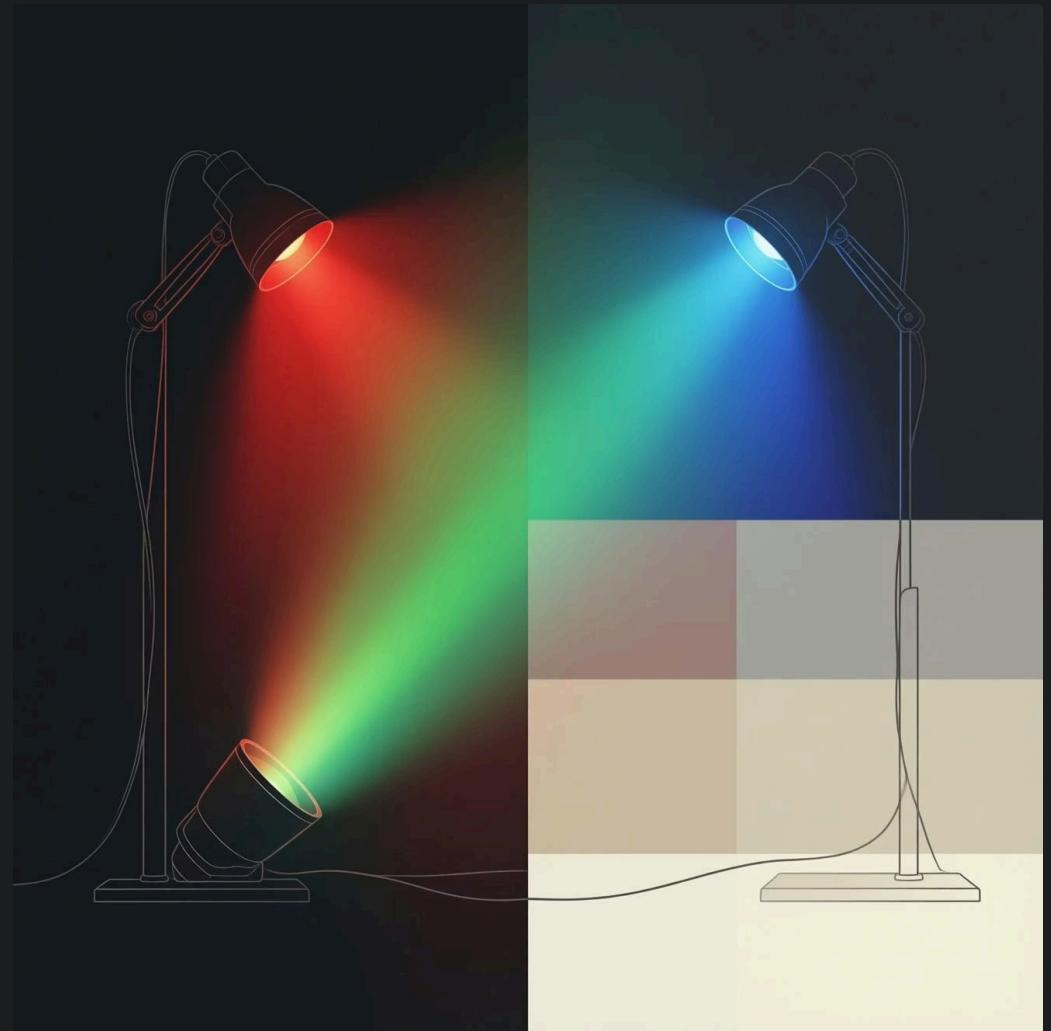
Renk Kodları: RGB Modeli

Dijital dünyadaki renklerin büyük çoğunluğu, **Red** (Kırmızı), **Green** (Yeşil) ve **Blue** (Mavi) olmak üzere üç ana rengin farklı oranlarda karıştırılmasıyla elde edilir. Bu sisteme **RGB Modeli** denir.

- Her renk kanalı (Kırmızı, Yeşil, Mavi) 0 ile 255 arasında bir değer alır.
- 0 değeri rengin hiç olmadığını, 255 ise rengin tam yoğunluğunu gösterir.

Örnekler:

- Saf Kırmızı: (255, 0, 0)
- Siyah: (0, 0, 0) - tüm renklerin yokluğu
- Beyaz: (255, 255, 255) - tüm renklerin maksimum yoğunluğu



Ses Verisinin Temsili

Ses, aslında havada yayılan analog bir dalgadır. Bilgisayarlar bu analog dalgayı doğrudan anlayamazlar. Dijital ortama aktarılabilmesi için sesin dönüştürülmesi gereklidir.

- Bu dönüşüm sürecine **örnekleme (sampling)** denir.
- Ses dalgasının yüksekliği belirli aralıklarla ölçülüp sayısal değerlere dönüştürülür.
- Örnekleme sıklığı ve her bir örneğin hassasiyeti (bit derinliği), dijital sesin kalitesini belirler.

Böylece, kulağımıza gelen melodi veya konuşma, bilgisayar için bir dizi sayıya dönüşür ve bu sayılar depolanır, işlenir ve çalınır.



Veri Büyüklüğü Sorunu: Görsel Bir Bakış

Günümüz dünyasında dijital veriler inanılmaz boyutlara ulaşıyor. Her gün çektiğimiz yüksek kaliteli fotoğraflardan, akıcı 4K filmlere kadar her şey, büyük miktarda depolama alanı ve bant genişliği gerektiriyor.



Yüksek Kaliteli Fotoğraflar

Her yüksek kaliteli fotoğraf, milyonlarca piksel içerir. Bu da her bir fotoğrafın devasa bir veri yiğini olmasına neden olur.



4K Filmler ve Video

4K çözünürlüklü bir film, saniyede 60 kare resim barındırır. Bu sürekli akan görsel şölen, veri boyutunu katlayarak artırır.

Bu Kadar Büyük Veriyle Nasıl Başa Çıkacağız?

Bu kadar büyük veriyi depolamak, aktarmak ve işlemek ciddi zorluklar yaratır. İşte bu noktada **veri sıkıştırma** devreye giriyor.



Depolama Zorlukları

Milyarlarca gigabaytlık veriyi güvenli ve erişilebilir bir şekilde saklamak, sürekli büyüyen bir altyapı maliyeti ve yönetim karmaçası demektir.



Aktarım Engelleri

Yüksek çözünürlüklü içerikleri internet üzerinden hızlı ve kesintisiz bir şekilde göndermek, mevcut ağ altyapıları için büyük bir yüktür.



İşlemci Yükü

Büyük boyutlu verilerin işlenmesi, bilgisayar işlemcileri üzerinde aşırı yük oluşturarak performansı düşürebilir ve enerji tüketimini artırabilir.

Neden Sıkıştırma Yaparız?

Veri sıkıştırma, yalnızca bir kolaylık değil, aynı zamanda dijital dünyamızın işleyişi için bir zorunluluktur. İşte sıkıştırmanın temel faydaları:



Depolama Alanı Tasarrufu

Sıkıştırma, sabit disklerinizde, bulut depolama alanlarınızda ve diğer depolama ortamlarınızda daha fazla veriye yer açar.



Veri Transfer Hızını Artırma

Daha küçük dosyalar, internet üzerinden daha hızlı aktarılır. Bu da indirme sürelerini kısaltır ve internet kotanızı korur.



İşlemci Yükünü Azaltma

Sıkıştırılmış veriler, işleme için daha az kaynak gerektirir. Bu, cihazların daha hızlı çalışmasına ve enerji verimliliğinin artmasına yardımcı olur.

Sıkıştırma Algoritmaları

Veri sıkıştırma algoritmaları, verilerin nasıl küçültüldüğüne dair farklı yaklaşımlar sunar. Temel olarak iki ana türü ayrırlırlar.



Sıkıştırma Türleri: Kayıplı ve Kayıpsız

Kayıplı Sıkıştırma (Lossy)

Bazı verilerin kalıcı olarak silinmesini içerir. Dosya boyutunu önemli ölçüde küçültürken, kalitede hafif bir düşüşe neden olabilir.



Kayıpsız Sıkıştırma (Lossless)

Hiçbir veriyi kaybetmeden dosya boyutunu küçültür. Sıkıştırılmış dosya, açıldığında orijinaliyle tamamen aynıdır.



Kayıplı Sıkıştırma: Detaylar Feda Edilir

Kayıplı sıkıştırma, insan duyularının fark edemeyeceği veya önemsiz kabul edilen bilgileri ortadan kaldırarak dosya boyutunu radikal bir şekilde düşürür.

İnsan Algısına Odaklı

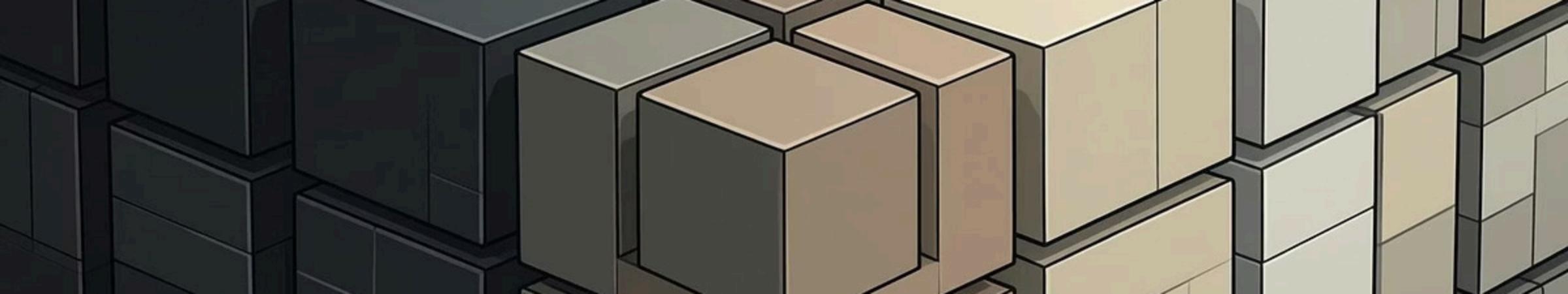
Göz veya kulak gibi insan algısının sınırlılıklarından yararlanır. Örneğin, bir resimdeki küçük renk farklılıklarını veya bir ses dosyasındaki yüksek frekanslı sesler çıkarılabilir.

Geri Dönüşü Yoktur

Bir kez sıkıştırıldıktan sonra, orijinal veriler geri getirilemez. Bu nedenle, kalitenin düşürülmesi kabul edilebilir olan durumlarda tercih edilir.

Yayın Kullanım Alanları

- JPEG (Resimler)
- MP3 (Ses)
- MP4 (Video)



Kayıpsız Sıkıştırma: Kusursuz Koruma

Kayıpsız sıkıştırma, bilginin hiçbir bölümünü feda etmeden dosyaları küçültür. Bu, verinin bütünlüğünün kritik olduğu durumlar için idealdir.

Veri Bütünlüğü

Sıkıştırma ve açma işlemleri sonrasında dosya, orijinal haliyle birebir aynıdır. Bu, kritik belgeler ve yazılım kodları için hayatı öneme sahiptir.

Değişmez Kalite

Kalitede herhangi bir düşüş olmaz. Her piksel, her karakter ve her ses dalgası korunur.

Önemli Uygulamalar

- ZIP, RAR (Genel dosyalar)
- PNG (Görseller)
- TXT (Metin belgeleri)

Projemde: RLE Algoritması

Bu projenin merkezinde, basit ama etkili bir kayıpsız sıkıştırma tekniği olan Run-Length Encoding (RLE) bulunmaktadır.

Run-Length Encoding (RLE)

RLE, tekrar eden veri dizilerini daha kısa bir formatta temsil ederek çalışır.

Özellikle tekrarlayan desenlere sahip veriler için oldukça verimlidir.

- Kayıpsız:** Hiçbir bilgi kaybı yaşanmaz, orijinal veri tamamen geri yüklenebilir.
- Basit ve Hızlı:** Uygulaması ve anaması kolay, işlem gücü gereksinimi düşüktür.
- Verimli:** Özellikle ardışık tekrarların sık olduğu grafikler ve basit resimler için idealdir.



İşte RLE kodlaması: "AAABBBCCCCDDDD" → "3A2B4C4D". Bu, verinin boyutunu %75 ile %80 azaltır.

RLE, veri akışındaki ardışık tekrarları tespit ederek sıkıştırma yapar. Nasıl çalıştığını bir örnekle açıklayalım:

RLE'nin Mantığı

Run-Length Encoding (RLE), veri akışındaki ardışık tekrarları tespit ederek sıkıştırma yapar. Nasıl çalıştığını bir örnekle açıklayalım:

Ardışık Karakter Tespiti

Veri içinde aynı karakterin art arda tekrar ettiği dizileri bulur. Örneğin, "AAAAAABCDGG".

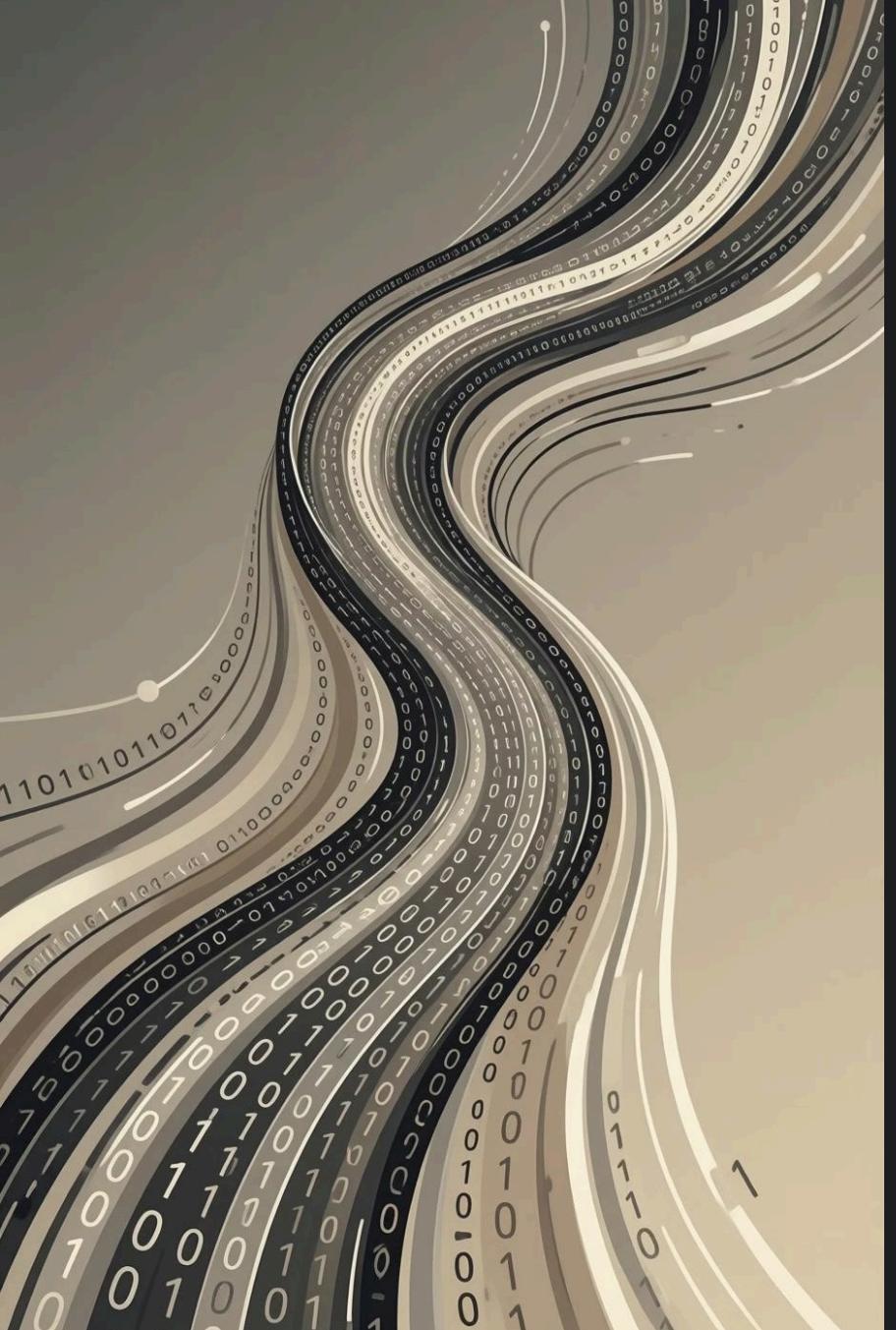
"Sayı + Değer" Forması

Tekrarlayan karakterleri "tekrar sayısı" ve "karakterin kendisi" şeklinde kodlar. "AAAAA" yerine "5A" yazar.

Dosya Boyutu Küçülmesi

Bu kodlama sayesinde, orijinal verinin boyutu küçülecek depolama ve aktarım verimliliği sağlanır.

Örnek: "AAAABBCDDD" verisi, RLE ile "4A2B1C3D" olarak kodlanır. Bu basit ama etkili yöntem, özellikle grafik tabanlı verilerde büyük avantaj sağlar.



BÖLÜM 4

RLE Detayları ve Python Projesi

Bu bölümde, Run-Length Encoding (RLE) algoritmasının ayrıntılarını inceleyeceğim ve Python ile geliştirdiğim RLE sıkıştırma/açma aracının özelliklerini ve çıktılarını keşfedeceğiz. Bilgisayar mühendisliği öğrencileri ve Python ile sıkıştırma algoritmalarına ilgi duyan herkes için bilgilendirici bir rehber niteliğindedir.

RLE Örneği: Basit Tekrar

Orijinal Veri

AAAAA (5 Karakter)

Sıkıştırılmış Veri

5A (2 Karakter)

Bu basit örnekte, ardışık beş 'A' karakteri sadece "5A" olarak temsil edilmiştir. Bu sayede, orijinal 5 karakterlik veri, sıkıştırılmış halde sadece 2 karakter yer kaplar. Bu, %60'luk etkileyici bir sıkıştırma oranı demektir.

RLE Örneği: Logo ve Çizgi Filmlerde Uygulama

Orjinal Veri

WWWWWWBBBBB

Bu örnek, RLE'nin özellikle görsel verilerde, örneğin logo tasarımlarında veya çizgi film animasyonlarında neden bu kadar etkili olduğunu göstermektedir. Aynı renk piksellerinin ardışık dizileri sıkıştırılarak dosya boyutları önemli ölçüde azaltılabilir.

Sıkıştırılmış Veri

5W5B

Bu tür verilerde genellikle geniş alanlar boyunca aynı rengin tekrarlandığı desenler bulunur, bu da RLE'nin verimli bir şekilde çalışması için ideal bir senaryo sunar.

RLE'nin Potansiyel Dezavantajı: Veri Genişlemesi

Eğer veri tekrar etmiyorsa, RLE boyutu büyütebilir!

1

Orijinal Veri

ABC (3 Karakter)

2

Sıkıştırılmış Veri

1A1B1C (6 Karakter)

Boyut arttı!

Yukarıdaki örnekte görüldüğü gibi, her karakterin yalnızca bir kez tekrarlandığı durumlarda RLE, veriyi sıkıştmak yerine genişlemesine neden olur. Bu nedenle, RLE genellikle fotoğraf gibi karmaşık ve yüksek varyasyonlu verilerde kullanılmaz. RLE, en çok birbirini tekrar eden ardışık karakterlerin olduğu verilerde verimli çalışır.

Python ile RLE Sıkıştırıcı Projesi

Bu ders kapsamında, bilgisayar mühendisliği eğitimimin bir parçası olarak Python programlama diliyle bir Run-Length Encoding (RLE) sıkıştırıcı geliştirdim.

- ❑ Bu proje, hem teorik bilgiyi przeće dökme hem de gerçek dünya uygulamalarına yönelik bir adım atma fırsatı sundu.

Geliştirdiğim program, metin tabanlı verileri etkili bir şekilde sıkıştırabilir ve sıkıştırılmış veriyi orijinal haline geri döndürebilir. Bu, sıkıştırma algoritmalarının temel prensiplerini anlamak ve Python'da string işleme becerilerini pekiştirmek için harika bir öğrenme deneyimi oldu.

Projenin Temel Özellikleri

1

Metin Girişi

Kullanıcıdan doğrudan metin verisi alır.

2

Veri Analizi ve Sıkıştırma

Giren metni analiz ederek en uygun RLE sıkıştırma formatını uygular.

3

Geri Çözme

Sıkıştırılmış veriyi hızlı ve doğru bir şekilde orijinal metne döndürür.

4

Başarı Oranı Hesaplama

Sıkıştırma sonrası elde edilen veri boyutunun orijinal boyuta oranını (%) hesaplar.

Bu özellikler, projenin hem işlevsel hem de öğretici yönünü güçlendirmektedir, kullanıcılarla sıkıştırma sürecinin her adımını görsel ve sayısal olarak takip etme imkanı sunar.

```

def rle_sikistir(metin):
    # Boş veri gelirse hata vermesin diye kontrol ediyoruz
    if not metin:
        return ""

    sonuc = ""
    sayac = 1

    # Metnin 2. harfinden başlayıp sonuna kadar gidiyoruz
    for i in range(1, len(metin)):
        # Eğer bu harf bir öncekiyle aynıysa sayacı arttır
        if metin[i] == metin[i-1]:
            sayac += 1
        else:
            # Farklı harfe geçince öncekini kaydet (Sayısı + Harf)
            sonuc += str(sayac) + metin[i-1]
            sayac = 1 # Sayacı sıfırla

    # Döngü bitince son kalan grubu da eklememiz lazım
    sonuc += str(sayac) + metin[-1]

    return sonuc

def rle_coz(sifreli_metin):
    cozulmus = ""
    i = 0

    # Tüm metni tariyoruz
    while i < len(sifreli_metin):
        sayi_yazisi = ""

        # Sayı kısımlarını buluyoruz (mesela 12A ise 12 yi alması için)
        while i < len(sifreli_metin) and sifreli_metin[i].isdigit():
            sayi_yazisi += sifreli_metin[i]
            i += 1

        sayi = int(sayi_yazisi)
        harf = sifreli_metin[i]

        # Harfi sayı kadar çarpıp ekliyoruz
        cozulmus += harf * sayi
        i += 1

    return cozulmus

def oran_hesapla(orijinal, sikismis):

```

Geliştirdiğim Koddan Görüntüler

Aşağıda, Python ile geliştirdiğim RLE sıkıştırıcı programın kod arayüzünden alınmış bir ekran görüntüsü yer almaktadır. Bu görüntü, kodun yapısını ve yorum satırlarını göstererek, programın nasıl çalıştığını dair genel bir fikir vermektedir.

Bu görsel, kodun temizliğini ve anlaşılabilirliğini vurgulamakta, böylece diğer geliştiricilerin de projeyi kolayca inceleyip anlayabilmesine olanak tanımaktadır.

--- RLE Projesi ---

Metni girin (Örn: AAAAABBB): Gooooorkem

Sıkışmış Hali: 1G5o1r1k1e1m

Geri Çözülmüş: Gooooorkem

Sıkıştırma Oranı: %-20.00

Durum: Başarılı, veri kayıbı yok.

Program Çıktısı: Canlı Bir Demo

RLE sıkıştırıcının nasıl çalıştığını daha iyi anlamak için, gerçek bir girdiyle elde edilen çıktıyı inceleyelim:

- **Girdi:** Gooooorkem
- **Çıktı:** 1G5o1r1k1e1m

Bu örnekte, 'o' harfinin beş kez tekrarlandığı "Gooooorkem" kelimesi, RLE algoritması sayesinde "1G5o1r1k1e1m" olarak sıkıştırılmıştır. Bu çıktı, her karakterin ardışık tekrar sayısıyla birlikte temsil edildiğini açıkça göstermektedir. Tekrar etmeyen karakterler (G, r, k, e, m) ise "1" ile birlikte gösterilir.

Projenin Kazanımları

Bu RLE projesi, sadece bir algoritma uygulamakla kalmayıp, aynı zamanda çeşitli alanlarda önemli kazanımlar elde etmemi sağladı:

Veri Seviyesi Anlayışı

Verilerin bit ve byte düzeyinde nasıl depolandığını ve işlendiğini derinlemesine kavradım.



Algoritmik Düşünme

Karmaşık problemleri adım adım çözme ve verimli algoritmalar tasarlama yeteneğimi geliştirdim.



Python Pratiği

Python'da string manipülasyonları, döngüler ve koşullu ifadeler gibi temel konular üzerinde yoğun pratik yaptım.

Bu proje, gelecekteki yazılım geliştirme çalışmaları için sağlam bir temel oluşturmuştur.

Kaynaklar

Bu sunumun ve RLE projesinin hazırlanmasında aşağıdaki kaynaklardan yararlanılmıştır:

- **Ders Kitabı:** Chapter 1.4 & 1.9 (Veri Sıkıştırma Temelleri)
- **Python Resmi Dokümantasyonu:** String metodları ve veri yapıları için başvuru kaynağı.
- **GitHub Reposu:** https://github.com/gorkemggunay-cloud/BLM101_25360859016_GorkemGunay