

VERİ DEPOLAMA VE SİKİŞTİRME ALGORİTMALARI

Ders: Bilgisayar Mühendisliğine Giriş

Bölüm: Bilgisayar Mühendisliği

Öğrenci No: 25360859311

Ad Soyad: Ramazan Eren Güzel

Grup: 2. Grup

Günümüzde metin, resim ve ses gibi dijital veriler çok büyük boyutlara ulaşabilmektedir. Bu verilerin depolanması ve iletilmesi hem maliyet hem de hız açısından önemli bir problem oluşturmaktadır. Bu nedenle veri sıkıştırma algoritmaları geliştirilmiş ve yaygın olarak kullanılmaya başlanmıştır.

Bu çalışmada veri sıkıştırma kavramı incelenmiş, temel sıkıştırma yöntemlerinden biri olan Run-Length Encoding (RLE) algoritması açıklanmış ve Python programlama dili kullanılarak örnek bir uygulaması gerçekleştirilmiştir.

DİJİTAL VERİLERİN BİT DÜZEYİNDE TEMSİLİ

Metin Verileri

Bilgisayarlarda metin verileri karakterlerin sayısal karşılıkları ile temsil edilir. Örneğin ASCII kodlama sisteminde her karakter 8 bit ile ifade edilir. Unicode (UTF-8) ise daha fazla dil ve sembolü desteklemek amacıyla geliştirilmiştir.

DİJİTAL VERİLERİN BİT DÜZEYİNDE TEMSİLİ

Resim Verileri

Resimler piksellerden oluşur. Her pikselin rengi belirli sayıda bit ile saklanır. Örneğin RGB renk modelinde bir piksel 24 bit (8 bit kırmızı, 8 bit yeşil, 8 bit mavi) ile temsil edili

DİJİTAL VERİLERİN BİT DÜZEYİNDE TEMSİLİ

Ses Verileri

Ses verileri analog sinyallerin belirli zaman aralıklarında örneklenmesiyle dijital ortama aktarılır. Örnekleme frekansı ve bit derinliği arttıkça ses kalitesi artar ancak veri boyutu da büyür.

VERİ SIKIŞTIRMA NEDEN GEREKLİDİR?

Veri sıkıştırmanın temel amaçları şunlardır:

- Depolama alanından tasarruf sağlamak
- Veri iletimini hızlandırmak
- İnternet bant genişliğini daha verimli kullanmak
- Depolama ve iletim maliyetlerini azaltmak

Bu nedenlerden dolayı veri sıkıştırma algoritmaları günümüzde büyük öneme sahiptir.

Kayıplı ve Kayıpsız Sıkıştırma:

Kayıplı sıkıştırmada veri geri açıldığında bire bir aynı olmaz.
kayıpsız sıkıştırmada ise veri tamamen eski haline
döner.RLE kayıpsızdır.

RUN-LENGTH ENCODING (RLE) ALGORİTMASI

Run-Length Encoding, ardışık tekrar eden karakterlerin sayısını tutarak veri boyutunu azaltmayı amaçlayan basit bir sıkıştırma algoritmasıdır. Özellikle tekrar oranı yüksek olan verilerde etkilidir.

RLE Algoritmasının Avantajları:

Basit , hızlı ve kolay uygulanabilir bir algoritmadır.

Orijinal veri:

AAAAABBBCCDAA

RLE ile sıkıştırılmış hali:

5A3B2C1D2A

Bu yöntemde her karakterin tekrar sayısı ve karakterin kendisi birlikte saklanır.

PYTHON UYGULAMASI

Bu projede RLE algoritması Python dili kullanılarak uygulanmıştır. Program iki temel fonksiyondan oluşmaktadır:

- Encode (Sıkıştırma)
- Decode (Açma)

Encode Fonksiyonu

```
def rle_encode(text):
    encoded = ""
    count = 1

    for i in range(1, len(text)):
        if text[i] == text[i - 1]:
            count += 1
        else:
            encoded += str(count) + text[i - 1]
            count = 1

    encoded += str(count) + text[-1]
    return encoded
```

Decode Fonksiyonu

```
def rle_decode(encoded):
    decoded = ""
    count = ""

    for char in encoded:
        if char.isdigit():
            count += char
        else:
            decoded += char * int(count)
            count = ""

    return decoded
```

Girdi:

AAAAABBBCCDAA

Encoded çıktı:

5A3B2C1D2A

Decoded çıktı:

AAAAABBBCCDAA

SIKİŞTIRMA ORANI HESÂPLAMA

Sıkıştırma oranı aşağıdaki formül ile hesaplanmıştır:

$$\text{compression_ratio} = (1 - \frac{\text{len(encoded)}}{\text{len(text)}}) * 100$$

Formülün açıklaması:

(Sıkıştırılmış uzunluk/Orjinal
Uzunluk)X100

SONUÇ:

Bu çalışmada veri sıkıştırma kavramı incelemi^ş ve Run-Length Encoding algoritması teorik ve pratik olarak ele alınmıştır. RLE algoritması basit yapısı sayesinde anlaşılması kolay bir yöntemdir. Ancak yalnızca tekrar eden verilerde verimli çalışmaktadır. Bu nedenle daha karmaşık veriler için farklı sıkıştırma algoritmalarına ihtiyaç duyulmaktadır.

KAYNAKÇA:

Bu sunumda ders kitabı ve ders notları kullanılmıştır.