Run-Length Encoding (RLE) Compression Algorithm

Muhammad Afif Faizi
22360859261
Bilgisayar Mühendisliği

İçindekiler

Veri Sıkıştırmaya Giriş

- Veri sıkıştırma, dijital verilerin boyutunu küçültür
- Sıkıştırma, depolama alanı ve bant genişliği tasarrufu sağlar
- Metin, görüntü, ses ve video dosyalarında yaygın olarak kullanılır
- İki ana sıkıştırma türü vardır
  - Lossless compression / Kayıpsız sıkıştırma
  - Lossy compression / Kayıplı sıkıştırma
- Bu proje, basit bir kayıpsız yöntem olan Run-Length Encoding (RLE) üzerine odaklanır

Veri Sıkıştırmanın Önemi

- Veri sıkıştırma, depolama alanı ihtiyacını azaltır
- Sıkıştırılmış veriler daha hızlı iletilebilir
- Ağ bant genişliği kullanımını azaltır
- Veri depolama sistemlerinde performansı artırır
- Modern bilgisayar sistemlerinde yaygın olarak kullanılır

Sıkıştırma Türleri

- Veri sıkıştırma iki ana türe ayrılır
- Kayıpsız Sıkıştırma
- Orijinal veri tamamen geri getirilebilir
- Metin ve veri dosyaları için kullanılır
- Kayıplı Sıkıştırma
- Bazı veriler kalıcı olarak kaybolur
- Görüntü, ses ve video dosyalarında kullanılır

RLE Kavramı

- Run-Length Encoding (RLE), basit bir kayıpsız sıkıştırma tekniğidir
- RLE, ardışık tekrar eden karakterleri sayı ve karakter ile temsil eder
- Format: sayı + karakter
- Çok sayıda tekrar içeren verilerde en iyi sonucu verir
- Basit metin ve görüntü sıkıştırmada yaygın olarak kullanılır

Örnek:
AAAABBBBCCDD → 4A4B2C2D

RLE Örneği

- RLE’nin tekrar eden karakterleri nasıl sıkıştırdığını gösterir
- Ardışık karakterler birlikte gruplandırılır
- Her grup, sayı + karakter şeklinde temsil edilir

Orijinal Metin: aaabbc
RLE Kodlama:3a2b1c

RLE Avantajları

- Basit ve anlaşılması kolaydır
- Programlama dilleri ile kolayca uygulanabilir
- Kodlama ve kod çözme işlemleri hızlıdır
- Çok sayıda tekrar içeren verilerde etkilidir
- Kayıpsız bir sıkıştırma yöntemidir

RLE Dezavantajları

- Az tekrar içeren veriler için etkili değildir
- Bazı durumlarda dosya boyutu artabilir
- Rastgele veya karmaşık veriler için uygun değildir
- Performans, giriş verisinin yapısına bağlıdır

Algoritma Özeti

- Run-Length Encoding iki ana adımda çalışır
  - Encoding step
    - Ardışık karakterleri sayar
    - Sayı + karakter formatına dönüştürür
  - Decoding step
    - Sayı değerlerini okur
    - Orijinal metni tekrar oluşturur
- Sıkıştırma oranı kodlama işleminden sonra hesaplanır

Encoding Mantığı

- Kodlama, giriş metnini soldan sağa okuyarak başlar
- Ardışık karakterleri saymak için bir sayaç kullanılır
- Mevcut karakter öncekiyle aynıysa sayaç artırılır
- Karakter değiştiğinde, sayı ve karakter sonuç stringine eklenir
- İşlem metnin sonuna kadar devam eder

Decoding Mantığı

- Kod çözme, kodlanmış metni okumayla başlar
- Sayılar, karakterin kaç kez tekrar edeceğini belirlemek için okunur
- Her sayı bir karakterle takip edilir
- Karakter, sayı kadar tekrarlanır ve sonuç stringine eklenir
- İşlem kodlanmış metnin sonuna kadar devam eder

Sıkıştırma Oranı Mantığı

- Sıkıştırma oranı, sıkıştırmanın ne kadar etkili olduğunu gösterir
- Yüksek yüzde = daha iyi sıkıştırma
- RLE performansını değerlendirmeye yardımcı olur

Formül

```
def compression_ratio(original, compressed):
    if len(original) == 0:
        return 0
    return (1 - len(compressed) / len(original)) * 100
```

Python Uygulaması Özeti

- Program Python dili ile uygulanmıştır
- Üç ana fonksiyon vardır:
    a. rle_encode(text)Metni RLE formatına dönüştürür
    b. rle_decode(text) Orijinal metni tekrar oluşturur
    c. compression_ratio(original, compressed) Sıkıştırma yüzdesini hesaplar
- Kullanıcıdan metin girişi alınır
- Çıktılar: encoded text, decoded text ve sıkıştırma oranı
- Farklı girişler için kolayca değiştirilebilir

rle_encode fonksiyonu

- Giriş metnini RLE formatına dönüştürür
- Ardışık karakterleri sayar
- ekrar eden karakterleri saymak için sayaç kullanır
- Karakter değiştiğinde sayıyı ve önceki karakteri sonuca ekler
- Metnin son karakterini ayrı olarak işler
- Çıktı sayı + karakter şeklindedir
- Simple and efficient for small text inputs / Küçük metinler için basit ve verimli
- Kod çözme ve sıkıştırma oranı hesaplamasının temelini oluşturur

```
Output
Input Giriniz: AAAABBBCCD
Encoding Result: 4A3B2C1D
```

rle_decode fonksiyonu

- Kodlanmış RLE formatından orijinal metni tekrar oluşturur
- Önce sayıları okuyarak tekrar sayısını belirler
- Her sayı bir karakterle takip edilir
- Karakteri sayı kadar tekrar edip sonuca ekler
- Çok basamaklı sayıları doğru şekilde işler
- Kod çözme işlemi basit ve hızlıdır

```
Encoding Result: 4A3B2C1D
Decoding Result: AAAABBBCCD
```

Sıkıştırma Oranı Örneği

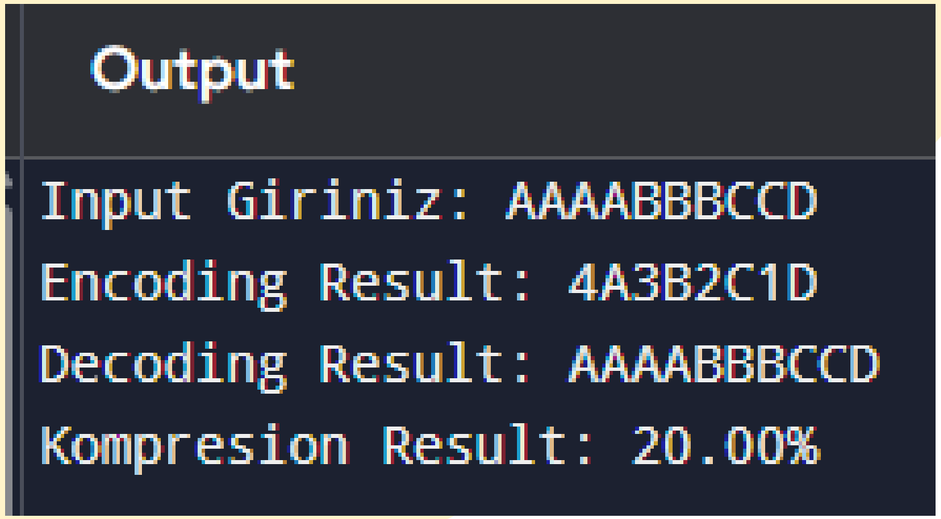
- Kodlanmış veriden sıkıştırma oranının nasıl hesaplandığını gösterir
- Teoriyi (formül) gerçek program çıktısıyla bağlar
- RLE sıkıştırmasının etkinliğini pekiştirir
- Farklı girdiler için performansı değerlendirmeye yardımcı olur

Hesaplama

```
def compression_ratio(original, compressed):
    if len(original) == 0:
        return 0
return (1 - len(compressed) / len(original)) * 100
```

Program Demonstrasyonu

- Programın tamamının çalışmasını gösterir
- Kullanıcı metin girer, program encoded text, decoded text ve sıkıştırma oranı çıktısı verir
- Fonksiyonların birlikte nasıl çalıştığını görselleştirir
- Kodlama, kod çözme ve sıkıştırma hesaplamasının mantığını pekiştirir



Zorluklar ve Öğrenilenler

- Ardışık karakter sayımını doğru yapmak
- Kenar durumlarını yönetmek (boş string, tek karakter)
- Kod çözmenin orijinal metinle eşleşmesini sağlamak
- Sıkıştırma oranını doğru hesaplamak
- RLE’nin ne zaman etkili olduğunu anlamak
- Yorum satırları ve okunabilir kod yazmanın önemi
- Projeyi düzgün dokümante etmenin ve sunmanın önemi

User Input
↓
rle_encode(text)
↓
Encoded Text
↓
rle_decode(encoded_text)
↓
Decoded Text
↓
compression_ratio(original, encoded)
↓
Output Display

Program Akış Diyagramı

- Programın tamamının mantığını gösterir
- Sıralamayı görselleştirir: input → encode → decode → sıkıştırma oranı → çıktı
- Fonksiyonların nasıl etkileştiğini anlamaya yardımcı olur
- Teoriyi uygulama ile bağlar

Kenar Durum Örnekleri

- RLE programının dayanıklılığını gösterir
- Özel durumları hatasız işler
- Örnekler:
    a. Empty string → "" / Boş string
    b. Single character → "a" / Tek karakter
    c. Long repeated string → "aaaaaa" / Uzun tekrar eden string
- Farklı senaryolarda programın güvenilirliğini gösterir

Sonuç

- RLE, basit ve etkili bir kayıpsız sıkıştırma yöntemidir
- Python’da encode, decode ve sıkıştırma oranı fonksiyonları ile başarıyla uygulanmıştır
- Program normal durumları ve kenar durumlarını (boş string, tek karakter, uzun tekrarlar) doğru şekilde işler
- Kodlama ve kod çözme süreçlerini adım adım gösterir
- Sıkıştırma oranı hesaplaması, teoriyi gerçek program çıktısıyla bağlar
- Algoritma tasarımı, test ve dokümantasyonun önemi öğrenildi

İlginç Notlar

- RLE eski grafik formatlarında kullanılır
- Kayıpsız sıkıştırma tüm orijinal veriyi korur
- Algoritmanın karmaşıklığı O(n)
- Basit uygulanabilir, ancak temel sıkıştırma kavramlarını gösterir
- Sıkıştırma tekniklerini öğrenmek ve anlamak için harika

Ek Notlar

- Kodlama ve uygulamadaki zorlu kısımları vurgula
- Farklı girdileri test etme ve hata ayıklama ipuçları
- Kenar durumlarını doğru yönetmek programın güvenilirliğini artırır
- Temiz kod ve yorumların önemi öğrenildi
- Algoritmanın mantığını anlamak, büyütmeden önce çok önemlidir

Kaynaklar

- Python resmi dokümantasyonu – https://docs.python.org/3/
- RLE açıklaması – https://en.wikipedia.org/wiki/Run-length_encoding
- Python ile RLE online öğreticiler – https://www.geeksforgeeks.org/run-length-encoding/
- Kullanılan ders materyalleri