daire, simge, sembol, logo, amblem içeren bir resim

Açıklama otomatik olarak oluşturuldu

**BLM6106 - Veri Sıkıştırma**

**Projesi**

**Dersin Yürütücüsü:** Prof. Dr. Banu Diri

**Öğrencinin;**

**Adı – Soyadı:** Muhammed Kayra Bulut

**Numarası:** 23501059

**Ödev Konusu:** Json veya XML formatındaki dosyaları istatistiksel yöntemle sıkıştırma

**Seçtiğim Algoritma:** Huffman Algoritması (Yarı Dinamik Huffman Kodlaması)

**Kullandığım Kodlama Dili:** Python3.8

**Sıkıştırma Yapacağım Dosya Türü:** Json

**Yarı Dinamik Huffman Kodlaması Nedir?**

**Yarı Dinamik Huffman kodlaması**, istatistiksel veri sıkıştırma algoritmalarından biridir. Temel Huffman kodlaması statik bir şekilde çalışır; yani, tüm veri seti ilk önce taranır, frekanslar belirlenir ve bu frekanslara göre bir Huffman ağacı oluşturulur. Ancak yarı dinamik versiyonda işler biraz farklıdır.

**Yarı dinamik yaklaşımda iki ana aşama vardır:**

**Frekans Analizi:** Öncelikle veri seti (doküman, dosya, vb.) baştan sona taranır. Her bir sembol (genellikle bir karakter veya byte) için kullanım sıklığı (frekansı) hesaplanır. Bu, genellikle bir frekans tablosu veya dizisi kullanılarak yapılır.

**Huffman Ağacının Oluşturulması:** Frekans analizi tamamlandıktan sonra, bu frekanslara dayalı olarak bir Huffman ağacı oluşturulur. Bu ağaç, en sık kullanılan sembollerin daha kısa, en az kullanılan sembollerin ise daha uzun kodlara sahip olacak şekilde tasarlanır.

Bu iki aşamadan sonra, asıl veri seti bu oluşturulan Huffman ağacı kullanılarak kodlanır.

**Ben Bu Yönetmi Nasıl Kullandım?**

Temel olarak iki farklı yaklaşımla yaklaştım.

**Birinci Yaklaşım:** Json dosyasını okudum, okuduğum json dosyasının **her bir** karakterinin frekanslarını hesaplıyorum ve bunun yanında da **N** elemanlı karakter kümelerinin frekansını hesaplıyorum.

Sonrasında json içeriğinden **N** elemanlı karakter kümelerini teker teker çıkarıyorum. **N** elemanlı karakter kümeleri json içeriğinden çıktıktan sonra önceden frekanslarını oluşturduğum her **bir** elemanlı karakterler oluşan json içeriğinde geçmiyorsa bu karakterler gereksizdir. Bu karakterleri frekans dizisinden çıkarıyorum.

Gereksiz elemanları çıkarılmış olan frekans dizimle huffman ağacını oluşturuyorum ve encode işemini bu huffman ağacına göre yapmaya başlıyorum. Encode işlemi bittikten sonra header bilgisine huffman ağacını ekleyerek sıkıştırma işlemini gerçekleştirmiş oluyorum.

Sıkıştırdığım veriyi açmak için, sıkıştırdığım dosyanın header bilgisini okuyorum. Header bilgisine göre sıkıştırdığım dosyayı açıyor ve kaydediyorum.

**İkinci Yaklaşım:** Json dosyasını okudum, okuduğum json dosyasının **1’den N’e** kadar elemana sahip olan karakter dizilerinin frekanslarını hesaplıyorum.

Sonrasında json içeriğinden **N** elemanlı karakter kümelerinden başlayarak 2 elemanlı karakter kümelerine kadar, önce **N** elemanlı karakterleri çıkarıyorum. Sonrasında **N** elemanlı karakter kümeleri çıktığında, **N** elemandan daha az karakter kümelerinin json içeriğinde geçip geçmediğini teker teker kontrol ediyorum. Geçmeyenleri frekans dizisinden kaldırıyorum. Bunu sonrasında json verisinin ilk hali için **N-1, N-2, ... ,2** elemana kadar tekrar ediyorum. Bu yaptığım işlemin amacı header’da tuttuğum veri miktarını azaltmak.

Gereksiz elemanları çıkarılmış olan frekans dizimle huffman ağacını oluşturuyorum ve encode işemini bu huffman ağacına göre yapmaya başlıyorum. Encode işlemi bittikten sonra header bilgisine huffman ağacını ekleyerek sıkıştırma işlemini gerçekleştirmiş oluyorum.

Sıkıştırdığım veriyi açmak için, sıkıştırdığım dosyanın header bilgisini okuyorum. Header bilgisine göre sıkıştırdığım dosyayı açıyor ve kaydediyorum.

**Yöntemlerle Elde Ettiğim Başarımlar**

Yöntemlerin başarımlarını ölçmek için üç farklı test dosyası kullandım. Dosya isimleri **kucuk\_test.json (2.346 byte), orta\_test.json (14.273 byte), test.json (9.085.115 byte).**

**kucuk\_test.json** ve **orta\_test.json** dosyalarında çalışan bilgileri tutulmaktadır ve ingilizce karakterler kullanılmaktadır. Çalışan bilgileri id, isim, yaş, e-posta, telefon ve adres bilgilerini içermektedir. **test.json** dosyasındaysa yine çalışan bilgileri tutulmaktadır ama bu çalışan bilgileri çok ayrıntılı bir biçimdedir. Diğer bilgilere ek olarak url, hash vb. bilgiler de tutulmaktadır. Bunun yanında karakterler sadece ingilizceyle sınırlı değildir. Bunun yanında çince karakterler de yoğun bir biçimde bu dosyada mevcuttur.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dosya Adı** | **Orijinal Boyut** | **Birinci Yaklaşımla Sıkışmış Veri Boyutu** | **İkinci Yaklaşımla Sıkışmış Veri Boyutu** |
| **kucuk\_test.json** | 2.346 byte | 1.930 byte | 2.143 byte |
| **orta\_test.json** | 14.273 byte | 6.800 byte | 7.718 byte |
| **test.json** | 9.085.115byte | 5.935.255 byte | 7.345.789 byte |

**Boyut Değişim Tablosu**

Boyut değişim tablosunda kullanılan yöntemlerin **N** değerleri 1’den 10’a kadar ve **1** elemanlı karakter kümeleri haricindeki karakter kümelerinin frekans eşiği 1-1000 arasında değiştiğinde alınan en iyi performansı göstermektedir.

|  |  |  |
| --- | --- | --- |
| **Dosya Adı** | **Birinci Yaklaşımda En İyi Performansı Veren Frekans Eşiği / Karakter Boyutu** | **İkinci Yaklaşımda En İyi Performansı Veren Frekans Eşiği / Karakter Boyutu** |
| **kucuk\_test.json** | 1 / 2 | 91 / 2 |
| **orta\_test.json** | 1 / 2 | 626 / 2 |
| **test.json** | 1 / 2 | 1 / 2 |

**Frekans / Boyut Tablosu**

**(En İyi Performans Birçok Durumda Alınmıştır)**

**Frekans / Boyut Tablosu’na** baktığımızda aslında tüm yaklaşımların en iyi performansı karakter boyutu 2 olduğu zaman verdiğini görebiliyoruz. İkinci Yaklaşımdaysa frekans aralığının önemli bir ölçüt olduğunu anlayabiliyoruz.

|  |  |  |
| --- | --- | --- |
| **Dosya Adı** | **Birinci Yaklaşım** | **İkinci Yaklaşım** |
| **kucuk\_test.json** | %0,82 | %0,91 |
| **orta\_test.json** | %0,48 | %0,54 |
| **test.json** | %0,65 | %0,81 |

**Sıkıştırma Oranı Tablosu**

**Sıkıştırma Oranı Tablosu’na** baktığımızdaysa en iyi oran **orta\_test.json** dosyasında yakalanmıştır. Bunun sebebi dosyadaki karakter çeşitliliğinin **test.json** dosyasına göre az olması ve karakter sayısının **kucuk\_test.json** dosyasına göre fazla olmasıdır. Ayrınca birinci yaklaşımın her dosya için daha iyi performans verdiği gözlemlenmiştir.