BLM0334 Algoritma Analizi ve Tasarımı Projesi

2023-2024 Güz Yarıyılı

İBRAHİM AKSAN-BURAK ASLAN

Kullanılacak Algoritma: Huffman Algoritması

Konu Başlığı: **Huffman Algoritması Kullanarak Dosya Sıkıştırma İşlemi**

Problem:

Günümüzde artan veri boyutları çeşitli depolama sorunları yaratmaktadır. Özellikle sunucularda çok fazla veri depolanmakta ve aktarılmaktadır. Sunucularda veri depolama ve aktarımı sırasında verilerin hacmini azaltmak için veri sıkıştırma algoritmaları kullanmak gereklidir. Bu sorunun çözümü için Huffman algoritması kullanılmıştır.

Genel Proje Açıklaması:

Huffman algoritması, veri sıkıştırması alanında en çok kullanılan algoritmalardan biridir. Günümüzde birçok farklı uygulamada kullanılmaktadır.

Projenin Önemi:

Bu algoritma sayesinde, veriler daha az yer kaplayarak depolanabilir ve aktarılabilir. Bu da veri transferi hızını ve verimliliğini artırmaktadır.

Huffman algoritması, birçok farklı uygulamada kullanılmaktadır. Örneğin;

-Dosya sıkıştırma programları: Bu algoritma sayesinde, dosyalar daha az yer kaplayarak depolanabilir ve aktarılabilir.

-Arşivleme programları: Bu algoritma sayesinde, arşivler daha az yer kaplayarak depolanabilir ve aktarılabilir.

-Veri aktarımı: Bu algoritma sayesinde, veri transferi daha hızlı ve daha verimli bir şekilde yapılabilir.

-İletişim sistemleri: Bu algoritma sayesinde, iletişim kanallarının kapasitesi daha verimli bir şekilde kullanılabilir.

Proje Kapsamı:

Bu projede, Huffman algoritmasını kullanarak metin sıkıştırması yapacağız. Proje kapsamında, aşağıdakiler yapılacaktır:

-Huffman algoritmasının temelleri anlatılacak.

-Python ile Huffman algoritması kullanılarak metin sıkıştırma yapılacak.

-Sıkıştırılmış metnin performansı değerlendirilecek.

Projenin Literatürü:

1) **Application of Huffman Algorithm and Unary Codes for Text File Compression**

Metin verilerinin sıkıştırılması, teknolojik gelişmelerde önemli bir rol oynamaktadır ve bu süreç, veri aktarımı, kopyalama ve yedekleme gibi birçok kullanım alanını içermektedir. Veri güvenliği açısından kritik olan bu konu, Huffman algoritmaları gibi çeşitli sıkıştırma tekniklerini içermektedir. Bu teknikler, özellikle dijital aktörlerin önemli verilerini depolamak için kullanılmakta ve verilere yetkisiz erişimi önlemek amacıyla şifreleme olarak da kullanılabilmektedir. Huffman algoritması, orijinal verileri değiştirmeyen kayıpsız bir sıkıştırma tekniği sunmaktadır. Benzer şekilde, Unary Codes algoritması da frekanslarına göre sıralanarak verilerin sıkıştırılmasını sağlayan bir kayıpsız sıkıştırma tekniğidir. Bu tekniklerin birleşimi, dosya boyutunu küçültürken orijinal verilere geri dönebilme özelliği sunar. [1]

2) **Research on Image Compression Algorithm Based on Deflate**

Görüntü sıkıştırma, görüntüleri depolamak ve iletmek için kullanılan bir tekniktir. Görüntülerdeki gereksiz bilgileri ortadan kaldırarak, görüntülerin daha küçük boyutlarda olmasını sağlar. Görüntü sıkıştırma yöntemleri, kayıpsız ve kayıplı olmak üzere ikiye ayrılır. Kayıpsız yöntemler, görüntüdeki tüm bilgileri korur, kayıplı yöntemler ise bazı bilgileri atarak daha yüksek sıkıştırma oranı sağlar. Deflate, kayıpsız görüntü sıkıştırma için yaygın olarak kullanılan bir algoritmadır. LZ77 ve Huffman kodlama olmak üzere iki tekniği birleştirir. LZ77, verideki tekrarlanan desenleri, daha önceki oluşumlara yapılan referanslarla değiştirerek gereksiz bilgileri ortadan kaldırır. Huffman kodlama ise, sık kullanılan karakterlere daha kısa kodlar, az kullanılan karakterlere ise daha uzun kodlar atayarak veriyi daha da küçültür. Deflate algoritmasının etkinliği, sıkıştırılan veri türüne bağlıdır. Örneğin, metin dosyaları Huffman kodlama ile iyi bir şekilde sıkıştırılabilirken, resim dosyaları LZ77 ile daha iyi bir şekilde sıkıştırılabilir. Sonuç olarak, Deflate, çeşitli veri türleri için güçlü ve çok yönlü bir sıkıştırma algoritmasıdır. [2]

3) **Comparison of Huffman Algorithm and Lempel-Ziv Algorithm for audio, image and text compression**

Dijital iletişimde, verilerin daha hızlı ve güvenilir bir şekilde iletilebilmesi için sıkıştırılması gereklidir. Bu nedenle, veriler aktarım öncesinde kaynak kodlama olarak da bilinen veri sıkıştırma sürecinde daha az bit'e sıkıştırılmalıdır. Aynı zamanda, kaynak kodlama veri depolama için dosya boyutlarının sınırlandırılması açısından da önemlidir. En yaygın ve en çok kullanılan kaynak kodlama tekniklerinden ikisi Huffman Algoritması ve Lempel-Ziv Algoritmasıdır. Bu araştırmanın temel amacı, metin, görüntü ve ses sıkıştırma uygulamalarında hangi tekniğin daha iyi olduğunu belirlemektir. Her veri türünden dosyalar, analog-dijital çevirici ve pulse kod modülasyonu kullanılarak bit akışlarına dönüştürülmüştür. Bit akışları her iki sıkıştırma algoritması ile sıkıştırılmış ve her algoritmanın verimliliği, her veri türü için sıkıştırma oranını ölçerek hesaplanmıştır. [3]

4) **A Study on Data Compression Using Huffman Coding Algorithms**

Veri sıkıştırma, kodlama mekanizmalarının kullanıldığı, veri seti boyutunu azaltmak için uygulanan bir yöntemdir. Huffman kodlaması, özgün olarak metin sıkıştırmak için kullanılan başarılı bir sıkıştırma yöntemidir. Huffman'ın fikri, her sembol için sabit uzunlukta bir kod kullanmak yerine, kaynaktaki sıkça görülen bir karakteri daha kısa bir kod sözcüğü ile temsil etmek ve daha az sık görülen bir karakteri daha uzun bir kod sözcüğü ile temsil etmektir. [4]

5) An improved lossless image compression algorithm based on Huffman coding

Çalışmada, Run-Length Encoding (RLE), entropy coding ve sözlük tabanlı kodlama gibi mevcut kayıpsız sıkıştırma algoritmalarının tanıtımı yapılmıştır. Ardından, bu yöntemleri birleştiren popüler algoritmaların, Deflate, CALIC, JPEG-LS ve JPEG 2000'nin incelenmiş ve bazılarının avantajları ve dezavantajları belirtilmiştir. Renk indeksli görüntülerin kayıpsız sıkıştırılmasını artırmak için kullanılan palet sıralama yöntemleri de ele alınmıştır. Kayıpsız görüntü sıkıştırmada kullanılan çeşitli yöntemlerin eksikliklerine vurgu yapılarak, önerilen yeni bir algoritmanın tanıtımı yapılmıştır.[5]

Algoritma Analizi:

Zaman Karmaşıklığı:

Huffman algoritmasının zaman karmaşıklığı T(n) = n + nlogn şeklindedir. Bu karmaşıklık, algoritmanın her bir karakter için bir kez çalışmasını gerektirmesinden kaynaklanmaktadır. Algoritmanın ilk aşamasında, veri içerisindeki karakterlerin kullanım sıklığı hesaplanır. Bu işlem, O(n) karmaşıklığına sahiptir. İkinci aşamada, karakterler kullanım sıklıklarına göre sıralanır. Bu işlem, O(n log n) karmaşıklığına sahiptir. Sonuç olarak, Huffman algoritmasının toplam zaman karmaşıklığı, O(n log n) şeklindedir. Worst-case ve best-case durumlarında da karmaşıklık değişmeyecektir. Çünkü ne olursa olsun karakterler için sıralama ve ağaç oluşturma işlemi gerçekleşecektir. Çalışma hızı veri büyüklüğüne göre değişim gösterebilir.

Çalışma Hızı:

Huffman algoritmasının çalışma hızı, verilerin büyüklüğüne ve karakterlerin kullanım sıklığına bağlı olarak değişmektedir. Genellikle, veri içerisindeki karakterler eşit olarak dağılmışsa, algoritmanın çalışma hızı daha iyi olmaktadır. Bu durumun nedeni, eşit olarak dağılmış karakterler için algoritmanın ilk aşamasında karakterlerin kullanım sıklığını hesaplamak için daha az işlem yapmasıdır.

Performans:

Huffman algoritmasının performansı, sıkıştırma oranına göre değerlendirilebilir. Sıkıştırma oranı, sıkıştırılmış verinin orijinal veriye olan oranını ifade eder. Bu algoritma, veri içerisindeki karakterlerin kullanım sıklığına göre kodlar oluşturarak, sık kullanılan karakterler için daha kısa, az kullanılan karakterler için daha uzun kodlar oluşturur. Bu şekilde, sık kullanılan karakterlerin daha az yer kaplaması sağlanır.

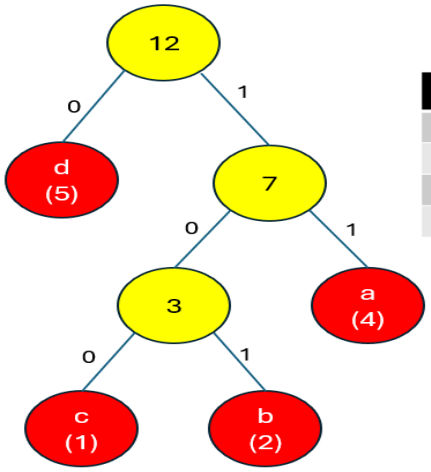
Sonuç:

Huffman algoritması basit ve etkili bir algoritmadır. Zaman karmaşıklığı ve çalışma hızı açısından oldukça verimlidir. Sıkıştırma oranı açısından da oldukça iyi performans sergilemektedir. Aşağıdaki tabloda sonuçlar daha net görülmektedir.

Algoritma Tasarımı:

Algoritma tasarımı kısmında zaman karmaşıklığını göz önünde bulundurarak bir tasarım gerçekleştirildi. En başta düğümleri oluşturabilmek adına bir Agac sınıfı oluşturuldu. Daha sonrasında Huffmann kodunu oluşturacağımız metnin harf frekansını elde etmek adına bir sayma algoritması yazıldı. Burdan çıkan sonucu da while döngüsü ile kurulan ağaç oluşturma kısmında kullanıldı. Her bir adımda agacin soluna ve sağına düğüm ekleyerek ağaç olusturuldu. Düğümler while döngüsü içerisinde toplanırken 'sorted' metodu ile tekrar sıralama yapıldı. Böylece ağaç döndürme işlemi yapılmış oldu.

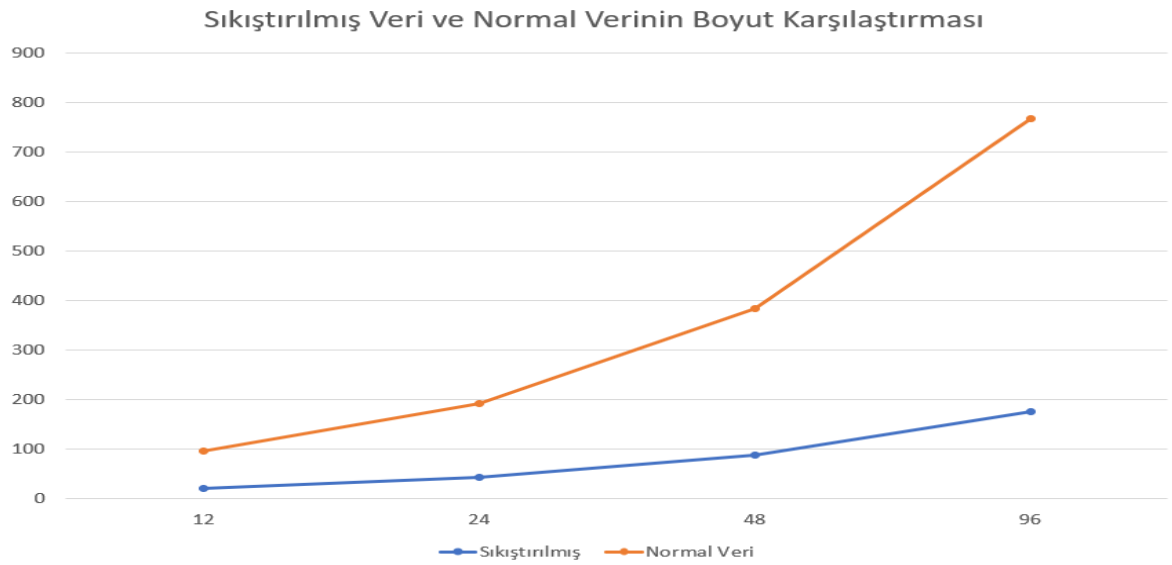
Ağacı çözmek için agaci_coz fonksiyonu kullanılmaktadır. Bu fonksiyon recursive olarak ilk olarak en sağa kadar gidecek eğer bir karakterle karşılaşırsa binaryStr adlı değişkene o karakteri ekleyecek ve oluşan stringi "karakter : string " şeklinde sozluk sözlüğüne atacaktır.



Metin:"aaaabbcddddd"

Harf	Frekans	Huffman Kodu	Bit Sayısı
d	5	0	1
a	4	11	2
b	2	101	3
c	1	100	3

Test ve Değerlendirme:



Algoritmada herhangi bir hata ile karşılaşılmadı. Bu bölümde yalnızca performans değerlendirilmesi ve sorunumuza çözüm bulup bulmadığı konusu irdelenecektir. Yukarıda

görüldüğü üzere programımızda kullandığımız metinde önemli ölçüde bir veri tasarrufu sağlanmıştır. Normal şartlar altında her karakter 1 bayt ile ifade edilecekken, Huffman algoritması ile az kullanılan karakterler çok bit ile, çok kullanılan karakterler ise az bit ile ifade edilerek boyutta azaltma sağlanılmıştır. Örneğin, yukarıdaki metin toplam 12 bayt yani 96 bit yer kaplayacaktı fakat Huffman kodlaması ile 22 bit yer kapladı. Yukarıdaki metin örneğinde %77 oranında daha az yer kaplandı. Sonuç olarak problemimiz de veri depolama konusu üzerine olduğundan Huffman algoritması problemimize çözüm sağlamıştır.

Sonuçlar:

- Algoritma analizi ve tasarımımız, kodumuz ile tutarlı bir ilişki gösterdi.
- Beklenen boyut ile hesaplanan boyut aynı bulundu.
- Recursive bir fonksiyon kullandığımızdan ötürü işlemcinin durumuna ve metindeki farklı karakter sayısına bağlı olarak sürede gecikmeler yaşandı.
- Karakterlerin eşit olarak dağılmışsa, algoritmanın çalışma hızı daha iyi olmaktadır. Bunun sebebi sıralama işleminin eşit frekanslı karakterler için daha kolay olmasıdır.
- Oluşturduğumuz while döngüsünde verimiz fazla olduğu zaman ağaç oluşturmak daha uzun süre aldı.

Projenin Zorlukları ve Katkıları:

Zorluk açısından; proje temelinde bir veri yapısı yattığı için kullanacağımız dili seçme konusunda zorluk yaşadık. Python'da algoritmayı implemente etmek bizi zorladı. Kullanmak İstediğimiz bazı modülleri kullanamadık. Fayda açısından; bir metin dosyasını Huffman kodlaması kullanarak nasıl sıkıştırabileceğimiz konusunda bilgi edindik. Algoritma analizi konu başlığının önemli bir kısmını kaplayan sıkıştırma algoritmaları konusunda tecrübe kazandık.

Kaynaklar:

[1] Wijaya, B. A., Siboro, S., Brutu, M., & Lase, Y. K. (2022). Application of Huffman algorithm and Unary Codes for Text File Compression. *Sinkron: jurnal dan penelitian teknik informatika*, *7*(3), 1000-1007.

[2] Liu, Y., & Zhao, L. (2023, July). Research on Image Compression Algorithm Based on Deflate. In *2023 International Conference on Data Science and Network Security (ICDSNS)* (pp. 01-05). IEEE.

[3] Bedruz, R. A., & Quiros, A. R. F. (2015, December). Comparison of Huffman Algorithm and Lempel-Ziv Algorithm for audio, image and text compression. In *2015 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)* (pp. 1-6). IEEE.

[4] Shoba, D. J., & Sivakumar, S. (2017). A Study on Data Compression Using Huffman Coding Algorithms. *International Journal of Computer Science Trends and Technology (IJCST)*, *5*(1), 58-63.

[5] Liu, X., An, P., Chen, Y., & Huang, X. (2022). An improved lossless image compression algorithm based on Huffman coding. *Multimedia Tools and Applications*, *81*(4), 4781-4795.