www.tubiad.org
ISSN:2148-3736
El-Cezerî Fen ve Mühendislik Dergisi
Cilt: 3, No: 2, 2016 (287-296)

El-Cezerî Journal of Science and Engineering
Vol: 3, No: 2, 2016 (287-296)

ECJSE

Makale / Research Paper

Huffman Algoritmasıyla Kayıpsız Hızlı Metin Sıkıştırma

Faruk BULUT

Gediz Üniversitesi, Müh. ve Mimarlık Fakültesi, Bilgisayar Müh. Bölümü, İZMİR
faruk.bulut@gediz.edu.tr

Geliş/Received: 04.04.2016 **Düzeltme/Revised:** 03.05.2016 **Kabul/Accepted:** 04.05.2016

Özet: Huffman Algoritması entropi kodlama yöntemiyle yapılan kayıpsız bir veri sıkıştırma yöntemidir. Bu yöntemde her karakter için ikilik sayı sistemde özel bir kod üretir. Veri içerisinde en çok kullanılan karakter için en kısa, en az kullanılan karakter için ise en uzun kod üretir. Huffman yönteminin uygulamasında kullanılan klasik yöntem ikili ağaç veri yapısıdır. Bu çalışmada Huffman yöntemi, önerilen bir yöntemle kodlanarak hesaplama süresi klasik yönteme göre azaltılmıştır. Bir matris tablo yardımıyla gerçekleştirilen kodlama uygulamalarında metin dosyaları bilinen Huffman kodlama yöntemine göre daha hızlı bir şekilde sıkıştırılabilmiştir.

Anahtar kelimeler: Huffman Coding, Hızlandırma, Metin Sıkıştırma

Fast Text Compression via Huffman Coding

Abstract: Huffman algorithm is a lossless data compression algorithm with the entropy coding method. This method generates a unique codeword for each character in the binary number system. In this technique, a short codeword is generated for most common characters and a long codeword is generated for less common characters in the text. In the classic implementation of Huffman coding a binary tree data structure is used. In this study, the computational time of the Huffman method is decreased with a proposed method. In the implementation using a matrix table, it could be carried out to compress text data faster than the well-known coding style of Huffman.

Keywords: Huffman Coding, Enhancement, Text Compression

1. Giriş

Veri büyüklüğün azaltılması işlemine veri sıkıştırma denir. Amaç kapasitenin yeterli olmasını sağlamak ve verinin bir ağ üzerinden daha kısa bir zamanda iletilmesine olanak sağlamak olabilir. Bilgisayardaki verileri kayıpsız (lossless) bir şekilde sıkıştırmak için çeşitli algoritmalar literatürde önerilmiştir. Huffman Coding, Lempel–Ziv (LZ), LZW (Lempel–Ziv–Welch), LZR (Lempel-Ziv–Renau) ve DEFLATE en bilinen yöntemlerden bir kaçıdır. Yüksek veri sıkıştırma oranına sahip bu tarz algoritmalardan bazıları olasılıksal tabanlı çalışmaktadır.

Veri sıkıştırma yöntemleri, farklı temellere ve farklı tipteki verilere göre farklı sonuçlar üretse de temelde hepsi "gereksiz (*redundant*) verileri yok etme" prensibine dayanmaktadır. Örneğin kelimelerde A harfine J harfinden daha çok rastlanmaktadır. Öyleyse rastlanma sıklığı yani frekansı çok olan harfi kısa kodla, az olan harfi ise uzun kodla kodlamak daha mantıklı olur. Böylece

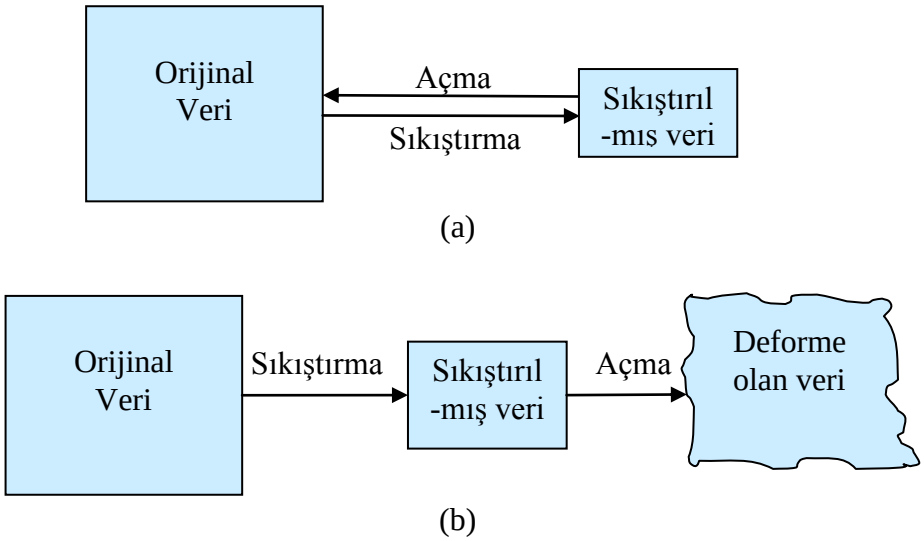
Bu makaleye atıf yapmak için
Bulut, F.,*"Huffman Algoritmasıyla Kayıpsız Hızlı Metin Sıkıştırma"* El-Cezerî Fen ve Mühendislik Dergisi 2016, 3(2); 287-296.

How to cite this article
Bulut, F., "*Fast Text Compression Via Huffman Coding*" El-Cezerî Journal of Science and Engineering, 2016, 3(2); 287-296.

alandan tasarruf sağlanmış olur. Veri sıkıştırma yöntemleri Kayıplı (*Lossy*) ve kayıpsız (*Lossless*) olmak üzere iki ana gruba ayrılır. Verilerin sıkıştırıldıktan sonra açıldıklarında orijinal veriyle ile aynı ise kayıpsız, farklı ise kayıplıdır.

Kayıpsız veri sıkıştırma yöntemi, sıkıştırılan verinin tekrar eski haline dönmesi gerektiğinde kullanılır. Örneğin metin tipindeki verilerde bu sıkıştırma yöntemi kullanılmalıdır. Çünkü bu tip veriler geri getirildiklerinde herhangi bir kelimede eksiklik olursa metni okumak ve anlamak zorlaşacak, hatta anlam kayıpları meydana gelecektir. Kısacası, metin belgeleri, kaynak kodları, çalıştırılabilir program dosyaları kayıpsız veri sıkıştırma yöntemiyle sıkıştırılmalıdır.

Kayıplı veri sıkıştırma yöntemlerinde, çıkartıldığında verinin en az düzeyde etkileneceği kısımlar çıkarılır, kalan veride de kayıpsız sıkıştırma yöntemi uygulanır. Herhangi bir veri kayıplı sıkıştırıldığında verinin tamamı değil, ancak bir kısmı geri getirilebilir. Kayıplı veri sıkıştırma yöntemlerinde verilerin çıkarılan parçaları insan gözünün ya da kulağının hissedemeyeceği kadardır. Bu yüzden bu yöntem fotoğraflar, video görüntüleri ve sesler için kullanılır. Genelde verilerin yüksek frekans değerlerini simgeleyen bölümleri veri eleme işlemine tabi tutulur çünkü insan gözü ve kulağı yüksek frekans değerlerini anlamada daha az hassasiyet gösterir. Kayıplı veri sıkıştırma algoritmaları JPEG ve MPEG türündeki dosyalarda uygulanır. Şekil 1'de gösterilen (a) olayında orijinal verinin kayıpsız bir şekilde sıkıştırılıp açılabildiği görülmektedir. (b) olayında ise kayıplı veri sıkıştırma örneği verilmiştir.



Şekil 1. Kayıpsız ve kayıplı sıkıştırma

Huffman algoritması aynı adı taşıyan bir araştırmacı tarafından önerilmiştir ve birçok alana uzun yıllardan beri uygulanmaktadır [1]. Bu alanda son yıllarda yapılan bazı çalışmalar vardır. Önerilen yöntemler hem teorik hem de pratik açıdan konuya yenilikler getirmiştir. Algoritmanın yeni türevleri literatürde tanımlandığı gibi farklı alanlara da uygulanmıştır. Görüntü işleme, video işleme, sinyal işleme ve metin sıkıştırma gibi birçok alana Huffman yöntemini uygulanmıştır. Bilindiği üzere JPG ve GIF gibi görüntü formatlarında kayıplı sıkıştırma ile alandan tasarruf edilmeye çalışılmaktadır. Pujar ve arkadaşları tarafından Huffman ile kayıpsız görüntü sıkıştırma yönteminin nasıl olması gerektiğini yaptıkları bir çalışmada anlatılmıştır [2]. Huffman ile görüntü sıkıştırma alanında daha hızlı çalışan bir algoritma *Hashing* yöntemi kullanılarak Reddy ve arkadaşları tarafından önerilmiştir [3]. 2015 yılında yapılan bir çalışmada ise kayıpsız veri sıkıştırma yöntemleri listelenerek algoritmik karmaşıklıkları, hesaplama süreleri, uygulanabilir alanları gibi açılar ele alınarak incelenmiştir [4]. Ayrıca farklı yöntemlerle performans artırımı üzerine önerilen bazı çalışmalar vardır [5], [6]. Bu çalışmaların hemen hepsi algoritmanın değişik

türevleri ile performans artırımını amaçlamaktadır ya da algoritmayı değişik alanlara uygulayarak daha iyi sonuçlar elde etmeyi hedeflemektedir.

Çalışmamızdaki ana hedef algoritmanın geliştirilmiş bir versiyonunu önermek ya da algoritmayı daha önce uygulanmamış bir alana uygulamak değildir. Amacımız algoritmanın kodlanmasında önerilen farklı bir veri yapısı ile kodlanarak hesaplama süresinin azaltılmasını sağlamaktır. Huffman yöntemi, bilinen şekli ile bağlı liste veri yapısı (*Linked List data structure*) kullanılarak kodlanmaktadır [7]. Bu yöntem algoritmayı bir miktar yavaşlatmaktadır. Bağlı liste yerine matris tablo kullanılarak yapılan uygulamada aynı sonuçların daha kısa bir sürede elde edildiği görülmüştür. Çalışmamız kayıpsız bir veri sıkıştırma algoritması olan Huffman tekniğinin hızlandırılmış bir modelini önermektir.

Çalışmamızın geriye kalan kısmı dört bölüm daha içermektedir. Takip eden ikinci bölümde Huffman yönteminin açıklamasına, üçüncü bölümde önerilen yönteme, dördüncü bölümde elde edilen deneysel sonuçlara ve son bölümde de değerlendirmelere yer verilmiştir.

2. Yöntem

Bu bölümde önce Huffman yönteminin açıklamasına ve önerilen yöntemin detaylarına yer verilmiştir.

2.1. Huffman Veri Sıkıştırma Algoritması

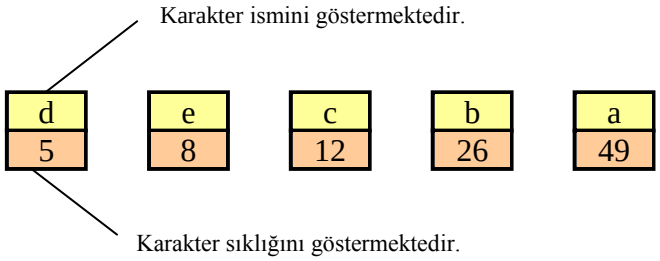
Bu yöntemde amaç, bir metinde frekansı yüksek sembolü veya karakteri kısa kodlarla, frekansı düşük sembolleri de uzun kodlarla temsil etmek ve bellekte tasarruf sağlamaktır. Örneğin bir metin dosyasında "a" ve "e" harflerine çok sık rastlanırken "j" harfine çok az rastlanır. Dolayısıyla "a" ve "e" harfi az bitle, "j" harfi daha fazla bitle kodlanır.

Bilgisayarda her bir karakter ASCII koduna göre 1 byte yani 8 bit uzunluğundadır. Örneğin 10 karakterlik bir katar (*string*) "aaaaaaaccf" şeklinde olsun. Bu katar 10 byte büyüklüğündedir ve 80 bit yapar. Bütün karakterleri 8 bitle kodlamak yerine sıklıklarına göre kodlayarak veriyi temsil etmek alan tasarrufu sağlar. Burada "a" karakterinin frekansı "f" karakterine göre fazladır. Buna göre "a" karakteri için "0" kodunu "f" karakteri için "10" kodunu, "c" karakteri için "11" kodunu kullanabiliriz. Böylece 10 karakterlik veriyi temsil etmek için, (a kodundaki bit sayısı)*(verideki a sayısı) + (c kodundaki bit sayısı)*(verideki c sayısı) + (f kodundaki bit sayısı)*(verideki f sayısı) = 1*7 + 2*2 + 2*1 = 12 bit gerekir. Bu durumda Huffman tekniği kullanarak %80 'in üzerinde bir kazanç elde etmiş olunur. Verideki karakter sayısına ve karakterlerin tekrarlanma sıklıklarına bağlı olarak Huffman algoritması çok yüksek bir oranda verim sağlayabilir.

Huffman tekniğinde semboller (karakterler) ASCII'de olduğu gibi sabit uzunluktaki kodlarla kodlanmazlar. Her bir sembol değişken sayıda uzunluktaki kod ile kodlanır. Bir veriyi Huffman algoritması ile sıkıştırabilmek için veride bulunan her karakterin ne sıklıkta tekrarlandığını bilmemiz gerekir. Mesela bir metin dosyasını sıkıştırıyorsak her bir karakterin metin içerisinde kaç kere geçtiğini bilmemiz gerekir. Her bir sembolün ne sıklıkta tekrarlandığını gösteren tabloya frekans tablosu denir. Dolayısıyla veri sıkıştırma işlemine başlamadan önce frekans tablosunu çıkarmamız gerekmektedir. Daha sonra yapılması gereken "Huffman Ağacı"nı oluşturmaktır. Huffman ağacı hangi karakterin hangi bitlerle temsil edileceğini (kodlanacağını) belirlememize yarar.

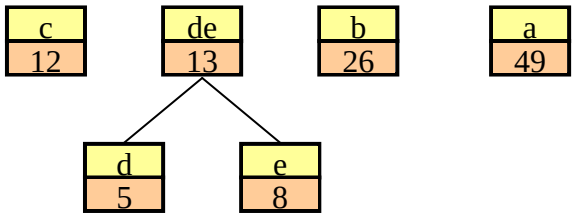
Sıkıştırılacak Metinde Geçen Semboller a, b, c, d ve e ve frekansları da sırasıyla 49, 26, 12, 5 ve 8 şeklinde olsun. Amaç her bir karakteri temsil edecek bit dizisinin bulunmasıdır. Bir Huffman Ağacı'nı, aşağıdaki adımları izleyerek oluşturabiliriz.

Adım 1. Öncelikle Huffman Ağacı'ndaki en alt düğümleri (yaprak) oluşturacak olan bütün sembolleri frekanslarına göre küçükten büyüğe doğru aşağıdaki şekildeki gibi sıralanır.



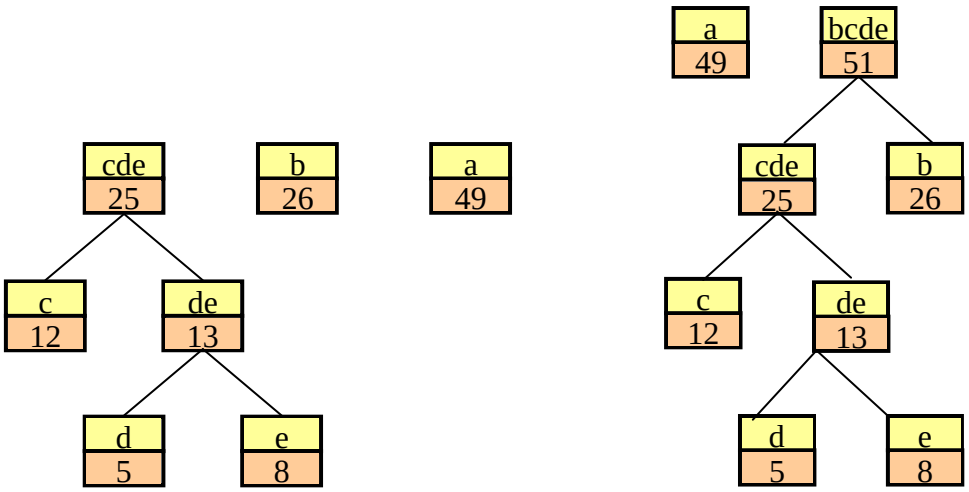
Şekil 2. Huffman Coding 1. aşama: sıralama

Adım 2. En küçük frekansa sahip olan iki sembolün frekanslarını toplayıp, yeni bir düğüm oluşturulur. Oluşturulan bu yeni düğüm, diğer var olan düğümler arasına –sıralamaya göre uygun yere- yerleştirilir. "d" ve "e" sembollerinin frekanslarını toplanıp, 13 frekansında yeni bir "de" düğümü elde edilir. Tekrar sıralama yapılarak 13 frekanslı yeni sembol "c" ve "b" sembolleri arasına yerleştirilir. "d" ve "e" düğümleri ise yeni oluşturulan düğümün yaprakları olarak kalır.



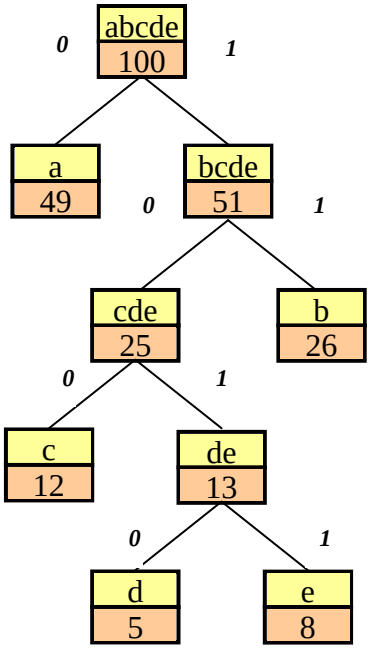
Şekil 3. Huffman Coding 2. aşama: birleştirme ve sıralama

Adım 3. İkinci adımdaki işlem tekrarlanır. Yani, en küçük frekanslı iki düğüm birleştirilir. Oluşan yeni düğümün frekansı 25 olacağı için, sıralamada "b" ve "a" düğümlerinin önüne geçer. İşlemler tekrar ettiğinde "bcde" birleşiminin frekansı 51 olur.



Şekil 4. Huffman Coding 3. Aşama

Adım 4. Ağaç veri yapısı tamamlandığında kök düğümün (root node) frekansı 100 aşağıdaki şekilde görüldüğü gibi olmaktadır. Ağacın her bir yaprağında görüldüğü üzere bir sembol vardır. Huffman Ağacı oluşturduktan sonra, her bir sembol belirli kodlarla işaretlenir. Sembol kodlama ağacının kök düğümünden başlanarak soldaki dala "0", sağdaki dala "1" bit değeri verilerek yapılır.



Şekil 5. Huffman Coding 5. Aşama

Her bir sembole kök düğümden başlayarak ulaşıldığında "0" ve "1" rakamlarının oluşan bir kombinasyon açığa çıkar. Örneğin "c" karakterine ulaşmak için önce "1" yolu, sonra "1" ve "0" yolu takip edilir. Bu durumda "c" için "110" kodu elde edilmiş olur. Bu şekilde bütün karakterlerin elde edilen Huffman kodları aşağıdaki çizelgede görülebilmektedir.

Huffman ile yapılan kodlamada ise 49 adet "a" karakteri için 49 * 1 = 49 bit, 26 adet "b" karakteri için 26 * 2 = 52 bit, 12 adet "c" karakteri için 12 * 3 = 36 bit, 5 adet "d" karakteri için 5 * 4 = 20 bit, 8 adet "e" karakteri için 8 * 4 = 32 bite ihtiyaç duyulur. Toplam gerekli olan bit sayısı 49 + 52 + 36 + 20 + 32 = 189'dir.

Çizelge 1. Karakterlerin Huffman Kodları

Sembol (Karakter)	Frekans	Kullanılan Bit Sayısı	Huffman Kodu
a	49	1	0
b	26	2	11
c	12	3	100
d	5	4	1010
e	8	4	1011

100 karakterlik bir metinde ASCII ile yapılan kodlamada 800 bit gereklidir. Fakat bir metin içerisinde bulunan *N* adet farklı karakter için en az ne kadar bit'e ihtiyaç duyulduğu entropi formülü ile hesaplanır [8]:

$$Bit\ sayısı = -\sum_{i=1}^{N} \log_2(P_i) \quad (1)$$

P_i her bir karakterin metin içerisindeki olasılığıdır. Yukarıdaki örnek için bu durumda 3 bit yeterlidir. $P_i = 5$ alınacağı için Bit sayısı 3 çıkacaktır. 300 bit (100×3) kullanmak yerine 36 bit ile yapılan kodlama ile veri sıkıştırma oranı ve kazancı şu şekilde bulunur [9]:

$$Sıkıştırma\ oranı = \frac{Sıkıştırılmış\ uzunluk}{Orijinal\ uzunluk} = \frac{36}{300} = 0{,}12 \quad (2)$$

$$Sıkıştırma\ kazancı = 1 - \ Sıkıştırma\ oranı = 1 - 0{,}12 = 0{,}88 \quad (3)$$

2.2. Hızlandırılmış Huffman Algoritması

Algoritmada bağlı liste veri yapısı kullanmak yerine iki boyutlu bir vektör yani matrisi kullanılmıştır. Yukarıda kullanılan örnek için bu tekniğe ait uygulamamın nasıl yapıldığını açıklayalım. Bu teknikte nesne yönelimli bir programlama dili kullanılarak iki boyutlu bir matris içeren bir sınıf (`class`) veri yapısı tanımlanır. Aşağıdaki çizelgede de görüldüğü üzere sınıf yapısının 1 ile isimlendirilen kısma frekansların toplamlarını, 2. ve 3. kısımlara birinci koordinatları, 4 ve 5. kısma ikinci koordinatlar yerleştirilir.

Çizelge 2. Sınıf (`class`) veri yapısı

1	2	3
	4	5

Bu koordinatlar topladığımız frekansların hangi koordinatlardan geldiğini belirtmektedir. Sınıf yapısından oluşan iki boyutlu bir matris dizi oluşturulur. Toplamda 5 adet karakter çeşidi olduğu için `A[5][5]`'lik iki boyutlu bir yeterlidir. Uygulanacak aşamalar şu şekildedir:

1. Oluşturulan yapının 1. kısımlarına harflerin frekanslarını yazılır.
2. Dizinin ilk satırında, 1. kısma yazılan frekanslar küçükten büyüğe doğru sıralanır.
3. Yapıların 2, 3, 4 ve 5. kısımlarına indisin dizideki koordinatları yazılır.
4. 1. satırdaki en küçük 2 frekansı toplanıp alt satırdaki yeni oluşturulan yapının 1.kısmına yazılır. Yapının 2 ve 3. kısmına frekansı küçük olan yapının koordinatları yazılır, 4 ve 5. kısma ise frekansı büyük olan yapının koordinatları yazılır.
5. Yeni oluşturulan yapı ve diğer yapıları küçükten büyüğe sıralanarak alt satıra yazılır.
6. Bu işlem tek bir yapı kalana kadar dek uygulanır.

Bu işlemler uygularken dikkat edilmesi gereken bir nokta vardır. Aşağıdaki matriste 2. satırdaki 12 ve 13 frekanslı yapılar toplandığında oluşan 25 frekanslı yapının 2., 3., 4. ve 5. kısımları, 12 ve 13 frekanslı yapıların 2., 3., 4. ve 5. kısmında yazan koordinatlar değildir. Bunlar 12 ve 13 frekanslı yapının kendi koordinatlarıdır. Yani 25 frekanslı yapının 2., 3., 4. ve 5. kısmına 0, 2 ve 0, 1 yazılmaz; 1, 0 ve 1, 1 yazılır. Yukarıdaki aşamaları uygularsak iki boyutlu sınıf dizisinin son hali aşağıdaki çizelge gibi olur.

Çizelge 3. Hızlandırılmış Huffman Algoritmasının iki boyutlu veri yapısına aktarılmış hali

	0			1			2			3			4		
0	5	0	0	8	0	1	12	0	2	26	0	3	49	0	4
		0	0		0	1		0	2		0	3		0	4
1	12	0	2	13	0	0	26	0	3	49	0	4			
		0	2		0	1		0	3		0	4			
2	25	1	0	26	0	3	49	0	4						
		1	1		0	3		0	4						
3	49	0	4	51	2	0									
		0	4		2	1									
4	100	3	0												
		3	1												

Oluşturulan tablodan bitler bulunur. Karakterlerin bitlerini bulmak için son satırdan özyinelemeli (*recursion*) olarak Huffman Coding ağacının yaprakları oluşturulur. Birinci koordinata giderken 0, ikinci koordinata giderken 1 değeri atanır. Ulaştığımız indisteki iki koordinat da birbirine eşitse ağacın sonuna gelindiği anlaşılır ve katarda tutulan 1 ve 0'lar yazdırılır. Bu şekilde tüm dizi gezilerek harfleri ifade etmek için hangi bit dizilimini kullandığımızı belirlenmiş olur. Algoritmamızın çıktısı harf ve Huffman kodu şeklinde Çizelge 4'de gösterilmiştir.

Çizelge 4. Hızlandırılmış Huffman ile elde edilen sonuçlar

Frekans	Sembol (Karakter)	Kullanılan Bit Sayısı	Huffman Kodu
49	a	1	0
26	b	2	11
12	c	3	100
5	d	4	1010
8	e	4	1011

Bu kodlama sistemi ile de aynı sonuçlar elde edilmektedir. Sonuç olarak 100 karakterlik bir metin dosyasını değişik kodlama biçimleriyle ifade edildiğinde ne kadar bite ihtiyaç duyulduğu Çizelge 5'e bakarak anlaşılmaktadır.

Çizelge 5. Kodlama Biçimleri ve Kullanılan Bit Sayıları

Kodlama biçimi	Kullanılması gereken en az bit Sayısı	
Unicode	100 * 16	1600
ASCII	100 * 8	800
3 bit ile	100 * 9	300
Huffman Coding	49+26*2+12*3+5*4+8*4	189

Eğer elimizde sıkıştırılmış bir veri dizisi ve bu dizinin frekans tablosu varsa, bu işlemlerin tersini yaparak orijinal veriyi elde edebiliriz. Şöyle ki, sıkıştırılmış verinin ilk biti alınır, eğer aldığımız bit bir kod sözcüğüne denk geliyorsa, o kod sözcüğünü denk gelen karakterin yerine koyarız. Eğer alınan bit bir kod sözcüğünün yerine denk gelmiyorsa, bir sonraki bitle beraber ele alıp denk gelen kod sözcüğü olup olmadığına bakarız. Bu işlemi, veri dizisinin sonuna kadar yaparız ve böylece Huffman kodunu çözmüş oluruz.

Sıkıştırılmış verinin 1 ve 0'lar ile ifade edilen kodunun sadece tek çeşit çözümü vardır. Bir kod dizisinden farklı semboller elde etmek imkânsızdır. Bu da kayıpsız veri sıkıştırmanın bir sonucudur.

3. Uygulama

Önerilen yöntem Pentium E2200 işlemcili, 2.2 GHz hızında, 1 GB RAM hafızaya sahip bir bilgisayarda denenmiştir. Linux OpenSuse 12.4 işletim sisteminde, C++ programlama dilini kullanarak GNU kütüphanesindeki "gcc" derleyicisi ile uygulamalar yapılmıştır. Hesaplama süresinin (*computational time*) hesaplanması için komut satırına aşağıdaki komut yazılmıştır:

```
>> gcc proje_kodu.cpp –o proje –lm –O2
>> time./proje
```

Çalışmada deneysel amaçlı olarak veri sıkıştırma algoritmaları için sıklıkla kullanılan önerilen bir külliyattaki (*corpus*) metin mesajları kullanılmıştır [10]. Bu külliyatta bulunan metin dosyaları çalışmada kullanılmıştır. Beşi gerçek üçü yapay olmak üzere toplam sekiz metin dosyasında algoritmaların hesaplama süreleri analiz edilmiştir ve aşağıdaki çizelgelere sonuçlar yerleştirilmiştir. Çalışmada kullanılmak üzere oluşturulan 1. Yapay Metin'de toplam 5 karakter vardır ve bu karakterlerin frekansları sırasıyla 1000000, 1000, 100, 10 ve 1'dir. 2. Yapay Metin'de toplam 8 karakter vardır ve frekansları sırasıyla 10000000, 1000000, 100000, 10000, 1000, 100, 10 ve 1'dir. Aynı şekilde 3. Yapay Metin'de toplam 8 karakter vardır ve frekansları sırasıyla 10000000, 2000000, 1000000, 10000, 2000, 1000, 200, 100, 70, 50, 20, 10 ve 1'dir. Klasik Huffman yönteminde Huffman Coding ağacı, bağlı listeler (*Linked List*) veri yapısı kullanılarak oluşturulmuştur.

Çizelge 6. Uygulama Sonuçları -1

Metin	Metindeki Karakter Sayısı	Karakter Çeşidi Sayısı	Klasik Huffman İle Çalışma Süresi	Hızlandırılmış Huffman İle Çalışma Süresi	Klasik Huffman İle Sıkıştırma Oranı	Hızlandırılmış Huffman İle Sıkıştırma Oranı
alice29.txt	152089	68	0.010 sn	0.003 sn	% 82.00	% 82.00
asyoulik.txt	125179	67	0.008 sn	0.003 sn	% 79.10	% 79.10
plrabn12.txt	481861	73	0.031 sn	0.006 sn	% 84.40	% 84.40
lcet10.txt	426754	85	0.021 sn	0.010 sn	% 81.30	% 81.30
alice29.txt	16.102	57	0.001 sn	0.001 sn	% 78.70	% 78.70
Yapay Metin 1.txt	1 milyon	5	2.01 sn	1.85 sn	% 66.63	% 66.63
Yapay Metin 2.txt	1 milyar	8	13.34 sn	8.01 sn	% 97.22	% 97.22
Yapay Metin 3.txt	1.3 milyar	9	14.40 sn	8.30 sn	% 67.23	% 67.23

Çizelge 7. Uygulama Sonuçları – 2

	Klasik Huffman İle Sıkıştırma süresi (sn)	Hızlandırılmış Huffman İle Sıkıştırma süresi (sn)	Klasik Huffman İle Yüzde olarak geçen süre	Hızlandırılmış Huffman İle Yüzde olarak geçen süre
The Red Room.txt	0,0010	0,0010	100	100
Hansel And Gretel.txt	0,0010	0,0010	100	100
Yapay Metin 1.txt	2,0100	1,8500	100	92,04
Yapay Metin 2.txt	12,3400	8,0100	100	64,91
Yapay Metin 3.txt	13,4000	8,3000	100	61,94

6. ve 7. Çizelgelerde verilen veri sıkıştırma süreleri ölçümü sistem zamanına göredir. Değişik metin dosyaları iki farklı veri yapısı kullanılarak sıkıştırılmıştır ve aynı sonuçlar elde edilmiştir. Çalışmada önerilen yöntem hesaplama süresi açısından diğer yönteme göre daha iyi sonuçlar vermiştir. Görüldüğü üzere önerilen model ile bazı metin dosyalarında %61,94 oranında daha hızlı veri sıkıştırma oranı elde edilmiştir. Ayrıca gerçek ve yapay metinlerin sıkıştırılmasında benzer zaman kazancı bulunmaktadır.

4. Değerlendirme

Kayıpsız bir veri sıkıştırma yöntemi olan ve birçok alana uygulanabilen Huffman Algoritmasının aynı sonuçları veren hızlandırılmış bir biçimi bu çalışma ile önerilmiştir. Farklı bir veri yapısı modeli olarak matris bir tablo kullanılmış ve yapılan hesaplamalarda klasik yönteme göre aynı sonuçlar daha kısa bir sürede elde edilmiştir. Önerilen bu tekniğin uygulanabilirliği daha kolaydır ve alan karmaşıklığı açısından daha da avantajlıdır. Önerilen bu model bağlı listelerle yapılan uygulamalara göre daha kısa sürede sonuçlar vermesi açısından tercih edilebilir bir yapıdadır.

Kaynaklar

[1] HUFFMAN, David A., et al. A method for the construction of minimum-redundancy codes. Proceedings of the IRE, 1952, 40.9: 1098-1101.

[2] PUJAR, Jagadish H.; KADLASKAR, Lohit M. A New Lossless Method Of Image Compressıon And Decompressıon Using Huffman Coding Techniques. Journal of Theoretical & Applied Information Technology, 2010, 15.

[3] REDDY, T. Bhaskara, et al. A novel approach of lossless image compression using hashing and Huffman coding. In: *International Journal of Engineering Research and Technology*. ESRSA Publications, 2013.

[4] RAO, Smitha; BHAT, Pratima. Evaluation of lossless compression techniques. In: *Communications and Signal Processing (ICCSP), 2015 International Conference on*. IEEE, 2015. p. 1655-1659.

[5] LIN, Yih-Kai; HUANG, Shu-Chien; YANG, Cheng-Hsing. A fast algorithm for Huffman decoding based on a recursion Huffman tree. *Journal of Systems and Software*, 2012, 85.4: 974-980.

[6] CHUNG, Kuo-Liang; LIN, Yih-Kai. A novel memory-efficient Huffman decoding algorithm and its implementation. *Signal Processing*, 1997, 62.2: 207-213.

[7] MUNRO, Ian; NEKRICH, Yakov; VITTER, Jeffrey Scott. Dynamic data structures for document collections and graphs. In: *Proceedings of the 34th ACM Symposium on Principles of Database Systems*. ACM, 2015. p. 277-289.

[8] HASHEMIAN, Reza. Direct Huffman coding and decoding using the table of code-lengths. In: *Information Technology: Coding And Computing [Computers And Communications], 2003. Proceedings. ITCC 2003. International Conference On*. IEEE, 2003. p. 237-241.

[9] S. Korkmaz, Türkçe Metinlerin Statik Huffman Algoritması Kullanarak Sıkıtırılmasında Sıkıştırma Oranı Optimizasyonu, Konya: T.C. Selçuk Üniversitesi, 2003.

[10] ARNOLD, Ross; BELL, Tim. A corpus for the evaluation of lossless compression algorithms. In: *Data Compression Conference, 1997. DCC'97. Proceedings*. IEEE, 1997. p. 201-210.