Çukurova Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi, 34(4), ss. 9-19, Aralık 2019
Çukurova University Journal of the Faculty of Engineering and Architecture, 34(4), pp. 9-19, December 2019

Kanonik Huffman Benzeri Kodlama için Kod Sözcüklerinin Uzunluklarını Cebirsel Olarak Hesaplayan Bir Algoritma

Mustafa ORAL[*1], M. Mustafa AŞŞIK[1]

[1]*Çukurova Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Adana*

Geliş tarihi: 16.10.2019 Kabul tarihi: 20.12.2019

Öz

Kanonik Huffman kodları için gerekli olan kod uzunlukları iki aşamada üretilir. Bu makalede "prefix-free" özelliğine sahip değişken uzunluklu kanonik kodların üretilmesine temel olacak uzunlukları cebirsel yöntemle tek aşamada hesaplayacak bir algoritma önerilmektedir. Ancak, elde edilen kodların "sembol başına ortalama bit uzunluğu" genellikle optimum olmayıp, optimuma benzerlerinden daha yakındır. Kod uzunlukları, ağırlık dizisinin sıralı olması şartıyla, en sık kullanılan sembolden başlayarak hesaplanır. Önerilen algoritma; p_i, i. sembolün olasılığı ve e_i de kalan olasılıkların toplamı olmak üzere, kod uzunluklarını l_i= round ($\log(e_i/p_i)$) formülüne göre hesaplar. Son olarak, kanonik formdaki kodlar hesaplanan uzunluklardan elde edilir. Tüm süreç Θ(n) zamanda tamamlanır ve Θ(n) kelime uzunluğunda hafıza kullanılır.

Anahtar Kelimeler: Veri sıkıştırma, Kodlama, Huffman, Kanonik form, Prefix-free kodlar

An Algorithm that Calculates the Lengths of Codewords Algebraically for Canonical Huffman-like Encoding

Abstract

The lengths of codewords required for canonical Huffman codes are produced in two stages. An algorithm that calculate the lengths required for the production of prefix-free canonical codes in single stage by algebraic method is proposed in this paper. The "average bit length per symbol" of the resulting code is usually not optimal, but it is closer to optimal than similar ones. The lengths of the codewords are calculated starting from the most frequently used symbol, provided that the weight array is ordered. The proposed algorithm calculates the lengths of the codewords using the formula l_i= round ($\log(e_i/p_i)$) where p_i is the probability of i-th symbol and e_i is the sum of the remaining probabilities. Finally, the codes in the canonical form are obtained from the calculated lengths. The whole process is completed in Θ(n) time and uses Θ(n) words memory, where n is the number of symbols.

Keywords: Data compression, Encoding, Huffman algorithm, Canonical form, Prefix free codes

[*]Sorumlu yazar (Corresponding author): Mustafa ORAL, *moral@cu.edu.tr*

1. GİRİŞ

D. Huffman tarafından tasarlandığından beri, popülerliğini kaybetmeden birçok yıldır kullanımda olan Huffman Kodlama [1] farklı araştırmacıların üzerinde hala çalıştığı önemli bir algoritmadır. Huffman Kodlamanın performansını iyileştirmek ve geleneksel yoldan farklı bir şekilde kodları üretmek için değişik çalışmalar yürütülmektedir [2-6].

Basitliği ve hızı nedeniyle, Huffman kodlayıcılar birçok sıkıştırma alanında hala kullanılmaktadır. Görüntü sıkıştırması (JPEG), metin sıkıştırma (DEFLATE, GZIP gibi), ses sıkıştırması (MP3) Huffman kodlamanın önemli uygulama alanlarındandır [7]. Ayrıca, sıkışık veriyi çözmeden kod sözcüklerine rastgele erişebilme yeteneği, sıkışık veri yapıları ve sıkışık metin veri tabanları gibi birçok senaryo bu kodlamayı çekici kılmaktadır [8].

Kanonik Huffman kodları, Huffman kodlarının farklı bir yapısıdır. Aynı ortalama bit uzunluğuna sahip olmalarına rağmen Kanonik Huffman kodlarının, Huffman kodlara göre bazı avantajları vardır. Bu iki kodlama yapısından 2. Başlıkta bahsedilecektir.

Aritmetik kodlama, entropi tabanlı kodlayıcıların başka bir çeşididir. Aritmetik kodlayıcının sıkıştırma oranı Huffman kodlayıcıdan daha iyi olmasına rağmen, Huffman kodlayıcının hızı Aritmetik kodlayıcıdan daha iyidir. Ek olarak Huffman kodlamanın uygulanması Aritmetik kodlamadan daha kolaydır [9]. Bu nedenlerden dolayı, yerine göre, Huffman kodlayıcı Aritmetik kodlayıcıya göre tercih edilebilmektedir.

Huffman kodlarının yaygınlığı, hızı ve kolaylığı gibi nedenlerden dolayı, araştırmacılar farklı teknikler kullanarak Huffman kodlarını veya Huffman benzeri kodları üretmek için çeşitli çalışmalar yapmışlardır. Burada Huffman kodları ifadesi optimum "prefix free" özellikli kodlar anlamına gelirken, Huffman benzeri ifadesi optimuma yakın "prefix free" özellikli kodlar anlamına gelmektedir. "Prefix-free" ifadesi ise kullanılan kod sözcükleri kümesinde herhangi bir kod sözcüğünün başka bir kod sözcüğünün öneki olmadığı anlamına gelmektedir [10].

Moffat ve Katajainen, 1995 yılında, büyük alfabeler için optimal "prefix free" kodların uzunluklarını üretecek bir algoritma önermişlerdir [2]. Bu algoritma ile uzunluklar, iki aşamada üretilmektedir. Öncelikle, optimum Huffman ağacı için, iç düğümlerin derinlikleri ve daha sonra, bu derinlikler kullanılarak kod sözcüklerinin uzunlukları hesaplanır. Bu algoritmanın amacı "in-place" tekniği kullanarak hafızadan tasarruf etmektir. Sonuç olarak, ağırlık dizisinin sıralı olması şartıyla, Huffman kodlarını O(n) zamanında üretmek için O(n) yer kullanılır.

1997'de, Moffat ve Turpin sembol olasılıklarının tekrarına dayalı bir minumum artıklı kod (minumum redundancy code) tekniği önermişlerdir. Büyük alfabeler için önerilen bu algoritma $O(r+r \log n/r)$ zamanına ve hafıza yerine sahiptir. Burada r farklı frekans değerine sahip sembollerin sayısını, n ise sembol sayısını göstermektedir [3]. Bu çalışmayla ilgili olarak bir iyileştirmeden söz edilebilmesi için r değerinin n/2'den küçük olması gerektiği açıktır. Bu gereklilik büyük alfabeler için sağlanabilse bile küçük alfabeler için, r genellikle n/2'den büyüktür. Örneğin, Calgary Külliyatı için "paper2" dosyasında r=72 ve n=91, "bib" dosyasında r=79 ve n=81'dir.

Optimal "prefix free" kodlar için başka çalışmalar da yapılmıştır [8,11,12]. Ancak, bunlar veri sıkıştırmadan daha çok veri yapıları ve veri depolama alanlarında kullanılan alfabetik ikili ağaç yapılarıdır. Bu çalışmalarda, Kanonik Huffman kod sözcüklerinin uzunluklarını elde etmek için önce optimal Huffman ağacı tasarlanır ve daha sonra uzunluklar bu ağaçtan elde edilir. Bunlardan başka bir ağaç oluşturmadan da kod uzunluklarını hesaplayan algoritmalar önerilmiştir. Bununla birlikte, bu algoritmalar Huffman benzeri kodlar üretmektedir.

1999'da Graham Fyfee, Fyfee Kodlar olarak isimlendirdiği optimuma yakın ikili "prefix free" özellikli kodları [4], 2010'da Andrew Polar, Polar Kodlar olarak ta bilinen "Huffman olmayan ikili

ağaç" olarak isimlendirdiği başka bir optimuma yakın ikili "prefix free" özellikli kodları geliştirmişlerdir [5]. Polar kodların bağlam (context) tabanlı adaptif versiyonunun Google tarafından kullanıldığı iddia edilmektedir [5]. Fyfee ve Polar kodların yapım zamanı, 2. ve 3. Başlıklarda da anlatıldığı üzere, n adımdan sonra ayarlama gerektirmesi nedeniyle O(kn)'dir. Burada $k \geq 1$'dir.

2008 yılında ise, Dube ve Beaudoin, "atık önek kodları" (Disposal prefix Codes) olarak isimlendirdikleri bir başka algoritma ile optimal olmaktan ziyade "prefix free" kodların hızlı bir şekilde yapımını amaçlamışlardır. F herhangi bir sembolün frekansı ve $\Delta=F_{max}-F_{min}$ olmak üzere, kod ağacının yapım zamanı O (Δ+n)'dır. Söz konusu ağaçta, herhangi bir düğümde tek çocuğa da izin verildiğinden sembol başına ortalama bit uzunluğu benzerlerine göre optimumdan daha büyüktür [6].

Bu makalede ise, Kanonik "prefix free" özellikli kodlar için bir alfabedeki sembollerle ilişkili kod sözcüklerinin uzunluklarını cebirsel hesaplama yoluyla bulan bir algoritma önerilmektedir. Elde edilen ortalama bit uzunluğu çoğunlukla optimal değildir, ancak 6. Başlıkta da görüleceği üzere optimal değere, benzer algoritmalardan daha yakındır. Uzunlukları hesaplama zamanı ve kullanılan hafıza miktarı, ağırlık dizisinin sıralı olması şartıyla, Θ(n)'dir.

Bu makale şu şekilde düzenlenmiştir: Kanonik Huffman kodları 2. Başlıkta tanıtılmıştır. 3. Başlık Fyfee Kodları ve 4. Başlıkta Polar Kodları tanımlar. 5. Başlıkta, Huffman benzeri "prefix free" özellikli kod sözcüğü uzunluklarını cebirsel olarak hesaplayan algoritma sunulmaktadır. 6. Başlıkta test sonuçları analiz edilmekte, 7. Başlıkta da sonuçlar tartışılmaktadır. Makale boyunca "log" deyimi taban 2 logaritma olarak düşünülmelidir.

2. KANONİK HUFFMAN KODLAMA

Huffman Kodlama, en olası sembole en kısa kod sözcüğünü atama prensibine dayanmakta ve değişken uzunluklu "prefix free" özellikli kodlar sınıfına girmektedir. Huffman Kodlama ile elde edilen sembol başına ortalama bit uzunluğu, Aritmetik Kodlayıcıdan sonra entropiye en yakın ortalama değerdedir.

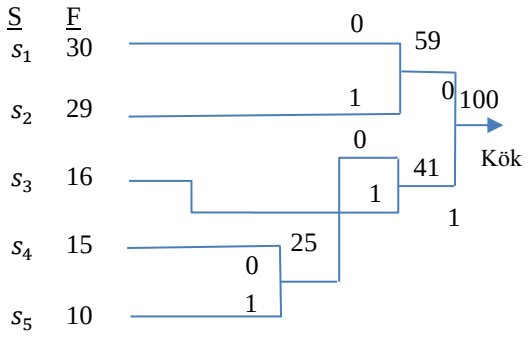
Bir mesaja ait sıralı bir frekans tablosu (veya olasılık dağılımı) verildiğinde, Huffman kodlama en düşük ağırlığa sahip iki sembolü birleştirerek kodlama sürecine başlar (Şekil 1). Böylece, oluşan yeni sembol tabloya eklenir. Tablo yeniden düzenlenir. Tekrar, tablodaki en düşük ağırlıklı iki sembol birleştirilir. Bu süreç, tablodaki tüm semboller tek sembol olarak birleştirilinceye kadar devam eder. Sürecin sonunda Huffman ağacı elde edilir. Huffman ağacının yapraklarında yer alan sembollerle ilişkili kod sözcükleri, kökten yapraklara doğru gidilerek bulunur.

Kanonik Huffman Kodları, Huffman kodlarının bir alt kümesidir. "Sayısal sıra özelliği" ne sahiptir. Sayısal sıra özelliği, aynı uzunluğa sahip kod sözcüklerini gösteren ikili değerlerin ardışık olduğu anlamına gelmektedir. Örneğin; "000", "001", "010", "011" gibi. Kanonik kodların bazı avantajları vardır: Uzunluk değerlerinin ardışık olmasından dolayı sıkıştırılmış dosya ile birlikte kod çözücüye gönderilmesi gereken "başlığın" boyutu önemli ölçüde azaltılır ve kullanılan alfabe ile birlikte uzunluk değerlerinin kod çözücüye gönderilmesi yeterlidir. Böylece kodlama ve kod çözme hızı artar ve kullanılan bellek miktarı da azalır [10,13,17].

Şekil 1 Huffman kodlarının ve Kanonik Huffman kodlarının elde edilmesini göstermektedir. Kanonik Huffman kod sözcükleri Huffman ağacından kod sözcüklerin uzunluklarına göre oluşturulur. Önce Huffman ağacı, sembol ağırlıklarına göre tasarlanır. Daha sonra ağaç, kökten sembolleri temsil eden yapraklara doğru gezilerek her sembole ait kod sözcüklerinin uzunlukları (başka bir ifade ile derinlik veya bit sayısı) tespit edilir. En küçük uzunluğa denk gelen kod sözcüğü her zaman bit olarak "0" dır. Eğer uzunluk 1'den büyükse, uzunluk değeri kadar sağ tarafa "0" ilave edilir (sola kaydırma). Örneğin uzunluk değeri 2 ise kod sözcüğü "00" dır. Bir sonraki kod sözcüğü, önceki kod sözcüğüne ikili düzende "1" ilave edilerek bulunur. Eğer

belirlenen kod sözcüğünün uzunluğu, karşı gelen uzunluk değerinden daha kısa ise fark değeri kadar kod sözcüğünün sağına "0" ilave edilir (sola kaydırma). Örneğin, uzunluk değeri 3 ise ve önceki kod sözcüğüne "1" ilave edilerek belirlenen kod sözcüğü "11" ise, "0" bit değeri kod sözcüğünün sağına ilave edilerek kod uzunluğu 3 yapılır ve yeni kod sözcüğü "110" olur. Şekil 1'de yer alan tabloda beş sembole sahip örnek bir dosya için kanonik kodlar yer almaktadır.

Semboller	Kod Sözcükleri	Uzunluklar	Kanonik Sözcükler
s_1	00	2	00
s_2	01	2	01
s_3	11	2	10
s_4	100	3	110
s_5	101	3	111



S: Semboller, F: Frekanslar

Şekil 1. Huffman ağacında kod sözcükleri ve uzunlukların belirlenmesi

Huffman kodlarının verimliliği, sembol başına ortalama bit uzunluğu ile ölçülür (A_v). Herhangi bir mesaj için A_v, $1 \leq i \leq n$ ve n kullanılan sembol sayısı olmak üzere, eğer i. sembolün kod uzunluğu l_i ve olasılığı p_i ise, sembol başına ortalama bit uzunluğu Eşitlik 1 ile tanımlanmaktadır [1].

$$A_v = \sum_{i=1}^{n} p_i \, l_i \qquad (1)$$

Huffman kodlamanın zaman karmaşıklığı; eğer ağırlık dizisi sıralı değilse O(nlogn), ağırlık dizisi sıralı ise O(n)' dir. Yer karmaşıklığı ise O(5n)'dir [2].

3. FYFEE KODLAR

Fyfee Kodların amacı, Huffman algoritması ile üretilen kod sözcüğü uzunluklarını hızlı bir şekilde tahmin etmektir. Herhangi bir mesaj için n sembolden oluşan bir alfabe $S=\{s_1, s_2, \dots, s_n\}$, sembollerin olasılık dağılımı $P=\{p_1, p_2, \dots, p_n\}$ ve $c_i \in \{0, 1\}$ olmak üzere kod sözcükleri kümesi $C=\{c_1, c_2, \dots, c_n\}$ olsun. l_i, c_i'nin uzunluğu ya da içerdiği bitlerin sayısıdır. L uzunluk kümesi ve $l_i \in$ L'dir. Herhangi bir kod sözcüğünün uzunluğu Eşitlik 2'de verilmiştir [18].

$$l_i = -\log(p_i) \qquad (2)$$

Ancak, l_i tamsayı olmak zorunda olduğundan Eşitlik 2 aşağıdaki gibi yazılabilir:

$$\lfloor -\log(p_i) \rfloor \leq l_i \leq \lceil -\log(p_i) \rceil$$

Ana fikir, başlangıçta her bir kod sözcüğünün uzunluğunu $\lceil -\log(p_i) \rceil$'ye ayarlamak ve

$$\sum_{i=1}^{n} 2^{-l_i} = 1 \qquad (3)$$

oluncaya kadar uzunluk değerini sırayla bir derece azaltmaktır. Burada Eşitlik 3 kod sözcüklerini temsil eden ikili kod ağacının tam bir ağaç olduğunu, yani iç düğümlerin ikişer çocukları olduğunu gösterir. Aynı zamanda kod sözcüklerinin tek olduğunu, diğer bir değişle "prefix-free" özelliğine sahip olduğunu da ifade eder (Kraft-MacMillan eşitsizliği: $\sum_{i=1}^{n} 2^{-l_i} \leq 1$).

Hangi l_i değerinin azaltılacağını belirlemek için bir R (artık değer) değişkeni tanımlanır (Eşitlik 4):

$$R = 1 - \sum_{i=1}^{n} 2^{-l_i} \qquad (4)$$

R<0 olması; kod ağacında eksik çocuklu düğüm olduğunu, R>0 olması da fazla çocuklu düğüm olduğunu gösterir. R=0 olduğu zaman tam bir kod sözcükleri kümesi elde ederiz. Hangi uzunluk değerini azaltacağımıza karar vermek için 2^{-l_i} ve R karşılaştırılır. Eğer $2^{-l_i} > R$ ise, herhangi bir işlem yapılmaz. Eğer $2^{-l_i} \leq R$ ise, l_i 1 eksiltilir.

S	P	L	2^{-l_i}
A	0,6	1	0,5
B	0,25	2	0,25
C	0,1	4	0,0625
D	0,05	5	0.03125

R= 0,15625

L	2^{-l_i}
1	0,5
2	0,25
3	0,125
5	0,03125

R= 0,09375

L	2^{-l_i}
1	0,5
2	0,25
3	0,125
4	0,0625

R= 0,0625

L	2^{-l_i}
1	0,5
2	0,25
3	0,125
3	0,125

R= 0

Şekil 2. Fyfee kod örneği

Basit bir örnekle işlemi anlatabiliriz. Sembol kümemiz (alfabe) S={A, B, C, D} ve olasılık dağılımı P={0,6, 0,25, 0,1, 0,05} olsun. Her bir s_i için l_i , 2^{-l_i} ve R hesaplanır. İşlem adımları Şekil 2'de gösterilmiştir. 1. adımda, her bir yukarı yuvarlanmış l_i değeri bulunur ve R, Eşitlik 3'e göre hesaplanır. En sık ağırlıklı sembolden başlayarak her 2^{-l_i} terimi R ile sırayla karşılaştırılır. 2^{-l_1} ve 2^{-l_2} değerleri R=0,15625'den büyüktür. 2^{-l_3} ise R'den küçük olduğu için l_3, 1 azaltılır. Yeni değerler 2. tabloda gösterilmiştir. 2. adımda 2^{-l_4} değeri güncellenmiş R'den küçüktür ve l_4, 1 azaltılır. Sonraki adımda 2^{-l_4} değeri R'ye eşit olduğundan tekrar 1 azaltılır. Artık R=0 olduğundan süreç sonlandırılır. Son L değerleri "prefix free" özellikli kod sözcüklerinin uzunluklarıdır. Sonraki işlem bu uzunluklardan kod sözcüklerinin kanonik formda elde edilmesidir.

4. POLAR KODLAR

Polar kodlar, Huffman kodlayıcı gibi veri sıkıştırma amacıyla kullanılan değişken uzunluklu ve "prefix-free" özellikli kodlardır. Polar kodların hesaplanması Fyfee kodlarınki gibi basittir. Uzunlukları elde etmek için olasılıklar yerine sembol frekansları kullanılır. Polar yöntemi şu şekilde çalışır: Alfabe ve frekansları $S=\{s_1, s_2, \dots, s_n\}$ ve $F=\{f_1, f_2, \dots, f_n\}$ olsun. Frekansların toplamı $T=\sum_{i=1}^{n} f_i$, l_i de kod sözcükleri kümesi $C=\{c_1, c_2, \dots, c_n\}$ den herhangi bir c_i kod sözcüğünün uzunluğu olsun. Tüm semboller, hâlihazırda frekanslarına göre sıralıdır. Önce frekanslar toplanır ve toplam T_1, 2'nin en yakın kuvvetine yukarı yuvarlanır. Her frekans değeri ise 2'nin en yakın kuvvetine aşağıya doğru yuvarlanır. Sıralanmış listede yukarıdan aşağıya doğru gidilerek ikinin kuvveti olarak yazılan frekans değerleri ikiyle çarpılır (katlama). İkiyle çarpma işlemi, frekansların toplam T_n değeri T_1 değerine eşit veya daha büyük oluncaya kadar devam eder. Eğer $T_1 = T_n$ ise, katlama işlemi durdurulur. Eğer T_n, T_1 den daha büyükse, $T_n = T_1$ oluncaya kadar ilk satırdan başlayarak tüm frekans değerleri sırayla ikiye bölünür. $T_n = T_1$ olduğu zaman işlem sonlandırılır. Bu durumda uzunluk Eşitlik 5'teki gibi hesaplanır:

$$l_i = \log T_n - \log F_i \tag{5}$$

F_i , son adımdaki i. frekans değeridir. 2. Başlıktaki örnekle Polar kodların oluşturulmasını açıklayalım. Alfabemiz S= {A, B, C, D}ve frekansları F= {60, 25, 10, 5} kümesidir. İşlem adımları Şekil 3'de gösterilmiştir. Öncelikle, frekanslar aşağıya doğru ikinin en yakın kuvvetine yuvarlanır. Frekansların ilk toplamı ise yukarı doğru ikinin en yakın kuvvetine ($T_1 = 128$) yuvarlanır. Sonra yuvarlanmış frekans değerleri

S	F	Yuvarlama
A	60	32
B	25	16
C	10	8
D	5	4
Total	100 (T)	128 (T1)

Katlama 1	Katlama 2	Ayar 1	Uzunluk
64	128	64	1
32	64	32	2
16	32	16	3
8	16	16	3
120 (Tn)	240 (Tn)	128 (Tn)	

Şekil 3. Polar kod örneği

ikiyle çarpılarak katlanır ve toplamları (T_n) hesaplanır. T_n=120<T_1 olduğu için katlama devam eder. Sonraki adımda T_n=240 olur ve katlama durdurulur. T_n, T_1'den daha büyük olduğu için T_n=T_1 oluncaya kadar son frekans değerleri sırayla ikiye bölünerek düşürülür. Gerekirse bölünme işlemi baştan başlayarak tekrar edilebilir. T_n=T_1 olduğu zaman kod sözcüklerinin uzunlukları Eşitlik 5'e göre hesaplanır. Sonraki adım uzunluklar kullanılarak kod sözcüklerinin bulunmasıdır.

Gerçekte, Polar kodlar ve Fyfee kodlar çok farklı algoritmalar değildir. Tek fark; Polar kodlar ağırlıkların logaritmasını aşağıya yuvarlarken, Fyfee kodlar yukarıya doğru yuvarlar. Ayar aşamasından önce, uzunluklardan R değeri hesaplandığında görülecektir ki, Fyfee kodlar için R>0 ve Polar kodlar için R<0'dır. Her ikisinde de amaç R değerini sıfırlamaktır.

5. KANONİK HUFFMAN KOD UZUNLUKLARININ CEBİRSEL YOLLA ELDESİ

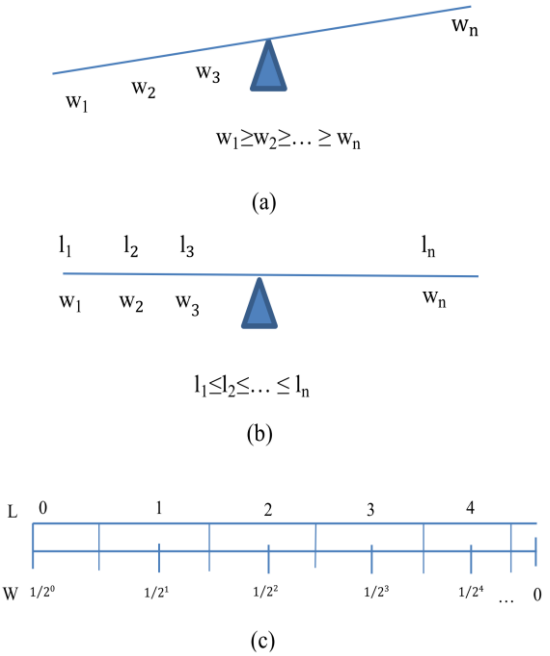
Bu makalede önerilen algoritma, geleneksel Huffman algoritmasının aksine, Fyffe ve Polar kodlar gibi yukarıdan aşağıya bir tasarıma sahiptir. Bununla birlikte, sembol başına ortalama bit uzunluğu optimum değere Fyffe ve Polar kodların sembol başına ortalama uzunluklarından daha yakındır.

Bu algoritmanın amacı optimum veya optimuma yakın "prefix free" özellikli kod uzunluklarını basit ve hızlı şekilde elde etmektir. Kullanılan ağırlık dizisinin sıralı olması şartı ile tüm süreç Θ(n) zamanda tamamlanır ve hafıza gereksinimi Θ(n) kelimedir. Burada n sembol sayısıdır. Bu noktadan sonra önerilen algoritma CKHK (**C**ebirsel **K**anonik **H**uffman **K**odlama) kısaltması ile anılacaktır.

5.1. CKHK Algoritması

Bir tahterevallimiz olduğunu var sayalım. Elimizdeki ağırlıkları, bu tahterevalliye sırayla bir uçtan bir uca yerleştirelim. Doğal olarak en ağır kısım aşağıya gelecektir (Şekil 4a). Amacımız bu tahterevalliyi dengeye veya dengeye en yakın konuma getirmektir. Bunu yapmak için, tahterevallinin en hafif ucuna daha ağır ve en ağır ucuna da daha hafif olmak üzere karşıt ağırlıklar yerleştiririz. Tahterevallimizdeki ağırlıklar, bir mesajdaki sembollerin frekansı veya olasılıklarıdır. Bu ağırlıkları dengelemek için kullanılan karşıt ağırlıklar ise kullanılan sembollere karşılık gelen kod sözcüklerin uzunluklarıdır (veya bit sayıları).

Bu durumda, en ağır kısma en kısa ve en hafif kısma en uzun uzunluk değeri karşılık gelmesi için uzunluk değerleri tahterevalli üzerinde sıralanmış olmalıdır (Şekil 4b). Problemimiz, tahterevalliyi dengeye veya dengeye en yakın konuma getirmek için kod sözcüklerinin uzunluklarının belirlenerek ağırlıklarla eşleştirilmesidir.



Şekil 4. Ağırlık ve uzunlukların eşleştirilmesi

Shannon'un Bilgi Teorisi (Information Theory) ve gözlemler bize ağırlıklar ve uzunluklar arasındaki ilişkiyi vermektedir (Şekil 4c). Elimizde bir alfabe ve sembollerin olasılık dağılımı olsun:

$S=\{s_1, s_2, \dots, s_n\}$, $P=\{p_1, p_2, \dots, p_n\}$. Burada n sembol sayısıdır ve P tahterevallimizdeki ağırlıklardır. Shannon, p_i olasılığına sahip s_i sembolü ile ilişkili kod sözcüğünün uzunluğunun, $1 \leq i \leq n$ olmak üzere, Eşitlik 2 ile hesaplanacağını belirtmiştir [18]. Ancak, uzunluk değeri olarak, l_i kesirli sayı olamayacağı için formül en yakın tamsayıya yuvarlanır. Bununla birlikte $p_i \geq 0{,}7$ değerleri için Eşitlik 1, $l_i \leq 0{,}51$ değerlerini üretir. Bu durumda l_i değeri, en yakın tamsayıya yuvarlanacağı için sıfır olmaması için doğrudan 1 alınır. Böylece aşağıdaki formül elde edilir (Y, en yakın tamsayıya yuvarlama fonksiyonu olmak üzere):

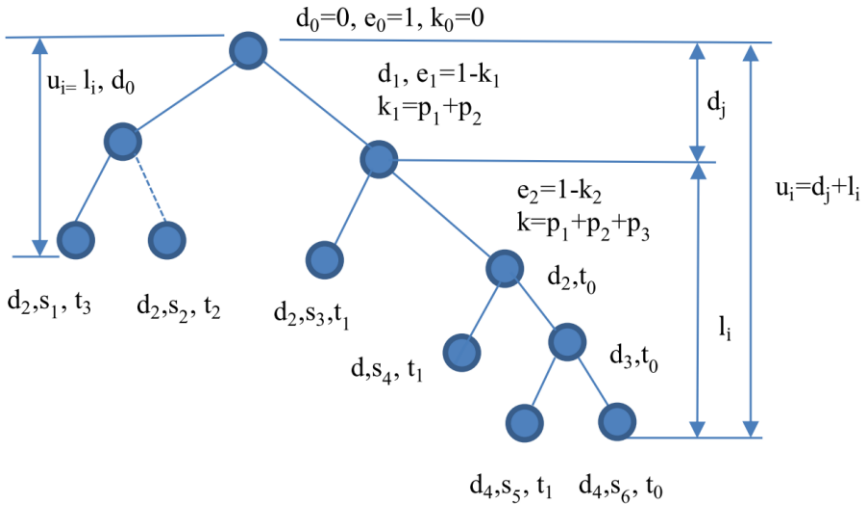
$$l_i = \begin{cases} 1 & \text{eğer } p_i \geq 0{,}7 \text{ ise} \\ Y(-\log(p_i)) & \text{aksi takdirde} \end{cases} \quad (6)$$

Gerçekte, kod uzunluklarını içeren uzunluk dizisi, ikili "prefix free" özellikli bir ağaca karşılık gelir. Tam bir ağaç yapısı elde etmek için Eşitlik 3'ün sağlanması gereklidir. Bundan dolayı bazı parametrelere ihtiyacımız olacaktır.

Süreç boyunca, p_i küçüldükçe, l_i'nin hesaplanmasında hata oluşur. Bu nedenle ölçeklendirme gerekli olur. Ölçeklendirme iki parametre kullanılarak yapılır: e_i ve k_i. k_i, işlem anına kadar hesaplanan olasılık değerlerinin kümülatif toplamıdır: $k_i = \sum_{x=1}^{i-1} p_x$. e_i ise, kalan olasılıkların toplamıdır: $e_i = 1-k_i$. Böylece ölçeklendirilmiş Eşitlik 6 aşağıdaki gibi formüle dönüşür:

$$l_i = \begin{cases} 1 & \text{eğer } p_i \geq 0{,}7 \text{ ise} \\ Y(\log(e_i / p_i)) & \text{aksi takdirde} \end{cases} \quad (7)$$

Parametreleri bir hayali ağaç üzerinde ele alırsak, işlem anında, e_i'nin ait olduğu düğüm, referans düğümdür. l_i, referans düğümden i. sembole olan uzaklıktır. Kökten i. sembole olan uzaklık ise u_i'dir. Kökten referans düğüme olan uzunluğa d_j diyelim. Eğer $d_j=0$ ise, referans düğüm köktür. Eğer $d_j \neq 0$ ise, $1 \leq j \leq n-1$ ve u_i, i. sembole karşılık gelen kod sözcüğünün gerçek uzunluğu olmak üzere, i. kod sözcüğünün uzunluğu, $u_i = d_j + l_i$ formülü ile verilir. N, d_j uzaklığındaki düğümlerin ve yaprakların sayısı olsun. Herhangi bir referans düğüme göre d_j uzaklığındaki yaprakların ve düğümlerin sayısı $N_j = 2^{l_i}$ formülü ile verilir. d_j uzaklığındaki a_i sembolünün konumu, $0 \leq m \leq N_j - 1$ olmak üzere, $t_m = (2^{u_i - u_{i-1}} * t_{m-1}) - 1$ eşitliği ile tanımlanır. Tüm parametreler, "prefix free" özellikli hayali bir ağaç yardımı ile Şekil 5'de gösterilmiştir.



Şekil 5. "prefix free" özellikli hayali bir ağaç üzerinde parametreler

e_i ve d_j parametrelerinin değerleri, d_m parametresinin önceki değerine göre değişir:

$$e_i=\begin{cases} e_i=1-k_{i-1} & \text{eğer } d_{m-1}\leq\frac{N_j}{2} \text{ ise} \\ e_i=e_{i-1} & \text{aksi takdirde} \end{cases}$$

$$d_j=\begin{cases} d_j=d_{j-1}+1 & \text{eğer } d_{m-1}\leq\frac{N_j}{2} \text{ ise} \\ d_j=d_{j-1} & \text{aksi takdirde} \end{cases}$$

s_n için, $e_n=1-k_{n-1}$ ve $e_n=p_n$ olması nedeni ile Eşitlik 7 sıfıra eşit olur. Bundan dolayı, u_n doğrudan d_j olarak alınabilir.

Böylece herhangi bir kod sözcüğünün uzunluğunu hesaplamak için gerekli tüm parametreler tanımlanmıştır. Uzunluklar elde edildikten sonra kod sözcükleri de kanonik formda kolaylıkla elde edilir. Örneğin, U=(1,2,3,3) ise, kod sözcükleri C=(0, 10,110,111) olur.

5.2. CKHK Sözde Kodu

Ağırlıkları verilen semboller ile ilişkilendirilmiş kod sözcüklerinin uzunluklarını hesaplayan algoritma aşağıda verilmiştir. Giriş verileri ağırlık dizisi ve toplam ağırlık değeridir. Ağırlık, olasılık dağılımı olarak verilirse toplam ağırlık değeri 1'dir. i. uzunluk hesaplandıktan sonra, ağırlık dizisinin i. pozisyonuna yazılır. Bu sayede, uzunlukları hesaplamak için n boyutlu bir dizi yeterlidir.

Girdi:

Sıralı ağırlık dizisi W, T_w = Toplam Ağırlık
Parametreler:

// U_ ifadesi önceki değeri gösterir.
E =T_w, K =0, D=0, U_=0, d=1, N=1, L=0, i=1;

Başla
Yap, W_i < n iken, // n alfabedeki sembol sayısı
{
Eğer d≤N/2 ise
E=T_w- K, D= D+1;
Eğer (W_i/E≥0,7 ve i<n) ise L = 1
Yoksa L = yuvarla$\left(\log(E/W_i)\right)$;
W_i= D+ L;

// ^ kuvvet işareti
d= [2^ (W_i- U_)*d]-1;
N=2 ^ L ;
K=K+W_i
i++ ;
}
Son
Çıktı:
W Uzunluk dizisi

Algoritma her durum için n adımda tamamlanır. Bu nedenle algoritmanın zaman karmaşıklığı Θ(n)'dir. Nadiren, algoritmanın çıktısının sırası bozulabilir. Herhangi bir l değerinden sonra l-1 gelebilir. Bu durumda, l ve l-1 yer değiştirilir. Bu değişikliğin ortalama bit uzunluğunda olumlu etkisi gözlenmiştir ve zaman karmaşıklığı üzerindeki etkisi ihmal edilebilir. Ağırlık dizisi kod sözcüklerinin uzunlukları için de kullanıldığı için yer karmaşıklığı Θ(n) kelimedir.

6. TEST SONUÇLARI

Fyffe kodlar, Polar kodlar, standart Huffman algoritması (SHA) ve CKHK sembol başına ortalama bit uzunluğu cinsinden karşılaştırılarak test edilmiştir. Her bir algoritma için, önce kod sözcükleri uzunlukları bulunmuş, sonrada Eşitlik 1 kullanılarak sembol başına ortalama bit uzunlukları hesaplanmıştır. Tüm hesaplamalar sıfır dereceli entropiye göredir.

Farklı tipte dosyalar içerdiği ve herkes tarafından kolayca erişebilindiği için Calgary Corpus test için seçilmiştir (http://corpus.canterbury.ac.nz/resources/calgary.tar.gz, Son erişim: 20.05.2019). Test sonuçları Çizelge 1'de gösterilmektedir.

Çizelge 1. Algoritmaların sembol başına bit uzunlukları

Calgary	Alphabet Size	Size (Byte)	Entropy	SHA	CKHK	Fyffe Codes	Polar Codes
Bib	82	111,261	5,200793	5,231822	**5,232128**	5,301720	5,327920
Book1	83	768,771	4,527168	4,561832	**4,562575**	4,647248	4,642746
Book2	97	610,856	4,792659	4,823421	**4,828855**	4,942489	4,893130
Paper1	96	53,161	4,983211	5,016911	**5,017023**	5,100655	5,112035
Paper2	92	82,199	4,601595	4,634246	**4,637324**	4,737263	4,679051
Paper3	85	46,526	4,665368	4,689986	**4,691942**	4,781030	4,740753
Paper4	81	13,286	4,700514	4,733350	**4,733725**	4,838865	4,826597
Paper5	92	11,954	4,936995	4,973651	**4,974822**	5,025094	4,979841
Paper6	94	38,105	5,009809	5,043799	**5,045925**	5,110586	5,131869
News	99	377,109	5,189671	5,227024	**5,229750**	5,303068	5,239286
Geo	257	102,400	5,646497	5,668656	**5,669730**	5,739534	5,679359
Obj1	257	21,504	5,948630	5,971774	**5,973076**	6,028924	5,984329
Obj2	257	246,814	6,260434	6,291299	**6,293139**	6,378133	6,363673
Pic	160	513,216	1,210213	1,660945	**1,661574**	1,668616	1,667565
Progc	93	39,611	5,199307	5,233919	**5,234348**	5,316975	5,282642
Progl	88	71,646	4,770264	4,799545	**4,799601**	4,828409	4,856616
Progp	90	49,379	4,869019	4,895200	**4,896861**	4,967477	4,943175
Trans	100	93,695	5,532914	5,568616	**5,569352**	5,632642	5,586194

SHA ile üretilen kodlar optimum kodlar olduğu için karşılaştırmalar, optimuma yakın kodlar üreten CKHK, Fyffe ve Polar kodlar arasında yapılmıştır. Sonuçlar, tüm dosyalar için optimum olan SHA değerlerine en yakın değerlerin CKHK olduğunu göstermektedir. Ayrıca SHA'dan sapma yüzdeleri (hata yüzdeleri) Çizelge 2'de gösterilmektedir. SHA ve CKHK arasındaki fark çok az olduğundan, bu algoritmaların hangisinin kullanılacağı kullanıcının seçimine bağlıdır (optimumluk, hız ve basitlik tercihi).

Çizelge 2. Algoritmaların SHA'ya göre hata yüzdeleri (%)

Calgary	CKHK-SHA	Fyfee-SHA	Polar-SHA
Bib	0,03	6,99	9,61
Book1	0,07	8,54	8,09
Book2	0,54	11,91	6,97
Paper1	0,01	8,37	9,51
Paper2	0,31	10,30	4,48
Paper3	0,20	9,10	5,08
Paper4	0,04	10,55	9,32
Paper5	0,12	5,14	0,62
Paper6	0,21	6,68	8,81
News	0,27	7,60	1,23
Geo	0,11	7,09	1,07
Obj1	0,13	5,72	1,26
Obj2	0,18	8,68	7,24
Pic	0,06	0,77	0,66
Progc	0,04	8,31	4,87
Progl	0,01	2,89	5,71
Progp	0,17	7,23	4,80
Trans	0,07	6,40	1,76

Polar kodlar ile bazı dosyalar sıkıştırılırken bazı sorunlarla karşılaşılmıştır. Örneğin, "book2" ve "news" dosyalarında $S_n >> S_{n-1}$ durumu oluşmuştur. Burada s, kullanılan alfabedeki herhangi bir sembol ve n, sembol sayısıdır. Bu durumda algoritma çözümsüz kalmakta, döngüye girmektedir. Diğer bir değişle R, sıfır olamamaktadır. Bu sorun, $S_n = S_{n-1}$ alınarak çözülmüştür. Çünkü tam bir ikili ağaç yapısında son iki eleman her zaman aynı olmak zorundadır. Diğer bir sorunla "pic" dosyasının sıkıştırılmasında karşılaşılmıştır. Alfabedeki bir sembol, örneğin s olsun, %87 olasılığa sahiptir. Bu durumda da, s ile ilişkili kod sözcüğünün uzunluğu sıfır olmaktadır. Dolayısı ile algoritma uygulanırken R=0 eşitliği sağlanamamaktadır. Bu sorunun çözümü s'ye ait kod sözcüğünün uzunluğu 1 alınarak bulunmuştur. Bu durumda olması gerekenin aksine R>0 olmuştur. Yani problem Fyffe algoritması ile çözülecek duruma dönüştürülmüştür. Söz konusu bahsedilen problemler ve çözümlerinden ilgi linkte [5] bahsedilmemektedir.

7. SONUÇ

Bu makalede, kanonik "prefix free" özellikli kodların yapımı için hızlı bir algoritma önerilmiştir. Bu algoritma ile "prefix-free" özellikli kanonik kod sözcüklerinin uzunlukları ağırlıklarına göre cebirsel hesaplama yoluyla bulunur. Daha sonra kod sözcükleri, bu uzunluklar kullanılarak kanonik formda elde edilir. Süreç $\Theta(n)$ zamanda tamamlanır ve $\Theta(n)$ kelimelik hafıza kullanır. Bununla birlikte, elde edilen sonuçlar genellikle optimum değil, optimuma çok yakındır. Çalışma sonuçları SHA ve CKHK arasındaki farkın benzerleri ile karşılaştırıldığında oldukça küçük olduğunu göstermiştir.

CKHK algoritması çok basittir ve uygulaması kolaydır. Hızı, basitliği ve kullandığı hafıza miktarı düşünüldüğünde, HTML sayfaları ve çevrim içi oyunlar gibi birçok uygulamada tercih edilebilir. CKHK, ayrıca hızın sıkıştırma oranından daha önemli olabileceği daha büyük alfabelerde de tercih edilebilir. Bu durumda, SHA ve CKHK arasındaki fark, sıkıştırılmış akış boyutu açısından ihmal edilebilir.

Bir sonraki çalışma olarak CKHK'nin adaptif versiyonunun geliştirilmesi planlanmaktadır. Bağlam (context) tabanlı adaptif CKHK uygulamasının hız ve sıkıştırma oranı açısından benzerlerine göre daha iyi sonuçlar elde edileceği düşünülmektedir.

8. KAYNAKLAR

1. Huffman, D., 1952. A Method for the Construction of Minimum Redundancy Codes, Proceedings of the IRE, 40(9), 1098-1101.

2. Moffat, A., Katajainen, J., 1995. In-place Calculation of Minimum-redundancy Codes, In: Akl S.G., Dehne F., Sack JR., Santoro N. (eds) Algorithms and Data Structures. WADS 1995. Lecture Notes in Computer Science, 955. Springer, Berlin, Heidelberg.

3. Moffat, A., Turpin, A., 1998. Efficient Construction of Minimum-redundancy Codes for Large Alphabets, in IEEE Transactions on Information Theory, 44(4), 1650-1657.

4. Geldreich, R., lzham-Fyffe Codes.wiki, https://code.google.com/archive/p/lzham/wikis/FyffeCodes.wiki, Son Erişim: 25.04.2019.

5. Polar, A., Non-Huffman Binary Tree, http://www.ezcodesample.com/prefixer/prefixer_article.html, Last Access 25.04.2019.

6. Dubé, D., Beaudoin, V., 2008. Fast Construction of Disposable Prefix-Free Codes, Comptes-rendus du International Colloquium on Signal Processing and its Applications, Kuala Lumpur, Malaisie, 49-54.

7. Hosseini, M., 2012. A Survey of Data Compression Algorithms and their Applications, 10.13140/2.1.4360.9924.

8. Navarro, G., Ordóñez, A., 2013. Compressing Huffman Models on Large Alphabets, In Proc. 23rd Data Compression Conference (DCC), 381–390.

9. Shahbahrami, A., Bahrampour, R., Rostami, M.S., Mobarhan, M.A., 2011. Evaluation of Huffman and Arithmetic Algorithms for Multimedia Compression Standards,

International Journal of Computer Science, Engineering and Applications (IJCSEA), 1(4),

10. Moffat, A., Turpin, A., 1997. On the Implementation of Minimum Redundancy Prefix Codes, in IEEE Transactions on Communications, 45(10), 1200-1207.

11. Leeuwen, J.Van., 1976. On the Construction of Huffman Trees, ICALP, 382-410.

12. Barbay, J., 2016. Optimal Prefix Free Codes with Partial Sorting, 27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016) 29, 1–13.

13. Chen, Y., Wan, G.C., Xia, Z.W., Tong, M.S., 2017. A Hardware Design Method for Canonical Huffman Code, 2017 Progress in Electromagnetics Research Symposium-Fall (PIERS - FALL), Singapore, 2212-2215.

14. Moffat, A., Witten, I.H., Bell, T.C., 1999. Managing Gigabytes: Compressing and Indexing Documents and Images, Second Edition, Academic Press, 36.

15. Connell, J.B., 1973. A Huffman-Shannon-Fano Code, in Proceedings of the IEEE, 61(7), 1046-1047.

16. Nekritch, Y., 2000. Byte-oriented Decoding of Canonical Huffman Codes, 2000 IEEE International Symposium on Information Theory (Cat. No.00CH37060), Sorrento, 371.

17. Chen, Y., Wan, G.C., Xia, Z.W., Tong, M.S., 2017. A Hardware Design Method for Canonical Huffman Code, Progress in Electromagnetics Research Symposium-Fall (PIERS - FALL), Singapore, 2212-2215.

18. Shannon, C.E., 1948. A Mathematical Theory of Communication, in the Bell System Technical Journal, 27(4), 623-656.