YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

TÜRKÇE'NİN BİÇİMBİLİM YAPISINA DAYALI BİR METİN SIKIŞTIRMA SİSTEMİ

Bilgisayar Yük.Müh.Banu Diri

F.B.E Bilgisayar Bilimleri ve Mühendisliği Anabilim Dalında Hazırlanan

DOKTORA TEZİ

Tez Savunma Tarihi	: 12 Kasım 1999
Tez Danışmanı	: Prof.Dr.M.Yahya KARSLIGİL (YTÜ)
Jüri Üyeleri	: Prof.Dr.Emre HARMANCI (İTÜ)
	: Prof.Dr.Selahattin KURU (BÜ)

İSTANBUL, 1999

İÇİNDEKİLER

SİMGE LİSTESİ

A	Kaynak alfabesi
A_b	Adres bilgisi
c	Zipf kanunu sabiti
cc	Kullanılan karakter sayısı
cf	Karakterin kullanım sıklığı
E	Ek ağacı ismi
f	Frekans
G	Gövde ağacı ismi
GS	Genişletme süresi
GS_{KeDDH}	Kelimeye Dayalı Dinamik Huffman yönteminin genişletme süresi
GS_{IHeDSH}	İlk Heceye Dayalı Dinamik Huffman yönteminin genişletme süresi
GS_{KeDDH}	Kelimeye Dayalı Dinamik Huffman yönteminin genişletme süresi
GS_{GKeDDH}	Geliştirilmiş Kelimeye Dayalı Dinamik Huffman yönt. genişletme süresi
H	Entropy
k	Kaynak mesajdan okunan miktar
L	Kraft-Mac Millan eşitsizliğinde kod uzunluğu
M	Tekrar sayısı
N	Eleman sayısı
P_i	Paketlenen kod sayısı
R	Redundancy (*tekrarlanma*)
r	Rank (*Kullanım sıklığı kademesi - Zipf kanunu*)
S	Ortalama bit sayısı
S_b	Sıkıştırma byte'ı
S_i	Sıkıştırma işaretcisi
SS	Sıkıştırma süresi
SS_{KeDDH}	Kelimeye Dayalı Dinamik Huffman yönteminin sıkıştırma süresi
SS_{IHeDSH}	İlk Heceye Dayalı Dinamik Huffman yönteminin sıkıştırma süresi
SS_{KeDDH}	Kelimeye Dayalı Dinamik Huffman yönteminin sıkıştırma süresi
SS_{GKeDDH}	Geliştirilmiş Kelimeye Dayalı Dinamik Huffman yönt. sıkıştırma süresi
T_s	Tekrar sayısı
T_k	Tekrar karakteri
t	Tekrarlanan değişik kelime sayısı(*Zipf kanunu*)
σ^2	Varyans
λ	Boş katar
η	Sıkıştırma verimi
ρ	Sıkıştırma oranı
ν	Sıkıştırma faktörü
β_i	a_i sembolünü kodlamak için gerekli bit sayısı
□	Boşluk

KISALTMA LİSTESİ

ARI	Arithmetic Coding
ATN	Augmented Transition Network(*Genişletilmiş Geçiş Ağı*)
ASCII	American National Standard Code for Information
BWT	Burrows-Wheeler Transform
CPU	Central Processing Unit (*Merkezi İşlem Birimi*)
EAS	Ek Adres Bilgisi
ESB	Ek Sıklık Bilgisi
EBCDIC	Extended Binary Coded Decimal Interchange Code
FGK	Faller-Gallager-Knuth
GKeDDH	Geliştirilmiş Kelimeye Dayalı Dinamik Huffman
HeDSH	Heceye Dayalı Statik Huffman
IHeDSH	İlk Heceye Dayalı Statik Huffman
KrDSH	Karaktere Dayalı Statik Huffman
KrDDH	Karaktere Dayalı Dinamik Huffman
KökDSH	Köke Dayalı Statik Huffman
KGAB	Kök-gövde Adres Bilgisi
KGSB	Kök-gövde Sıklık Bilgisi
KeDDH	Kelimeye Dayalı Dinamik Huffman
LZ	Lempel-Ziv
LZW	Lempel-Ziv Welch
MTF	Move-to-Front Coding
ÖYT	Örüntü Yerdeğiştirme Tekniği
RLE	Run-Length Encoding (*Ardışıl Karakter Sıkıştırma*)

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

TEŞEKKÜR

Doktora çalışmamda bilgisini, değerli görüşlerini ve desteğini benden esirgemeyen değerli hocam Sayın Prof.Mehmet Yahya Karslıgil'e çok teşekkür ederim.

Çalışmamın her aşamasında yaratıcı fikirleri, önerileri ve yardımlarıyla her zaman yanımda olan arkadaşım Arş.Grv.A.Gökhan Yavuz'a, tez kitabımın hazırlanması aşamasında daima destek olan arkadaşım Arş.Grv.Songül Albayrak'a, Y.T.Ü Bilgisayar Bilimleri ve Mühendisliği bölümündeki arkadaşlarım Dr.M.Elif Karslıgil, T.Ahmet Haktanır, Esin Ergün ve ayrıca Binbaşı Dr.Vedat Çoşkun'a teşekkür ederim.

Hayatım boyunca desteklerini her zaman hissettiğim sevgili eşim, annem ,babam ve kardeşlerime teşekkür ederim.

ÖZET

Bu doktora çalışmasında veri sıkıştırma konusunda yapılan diğer çalışmalardan farklı olarak, Türkçe metinlerin biçimbilimsel (morfolojik) olarak incelenmesi yapılmış ve bu inceleme sonucunda elde edilen gövde-kök, hece ve eklere ait istatistiksel verilere göre, yeni bir veri sıkıştırma yöntemi geliştirilmiştir. Sistemin başarımı ve çalışması değişik Türkçe metinlere uygulanarak değerlendirilmiştir.

Geliştirilen sistemin en önemli özelliği, var olan sıkıştırma yöntemlerinden farklı olarak, sıkıştırılacak veriyi ikili bilgi yapısında değil, Türkçe dilinin yapısına uygun şekilde hece, gövde-kök ve eklerine ayırarak değerlendirmesidir. Geliştirilen bu sıkıştırma yönteminde Huffman kodlama ağacı temel alınıp ilk olarak kelimenin heceleri, ikinci olarak kelimenin kök ve ekleri, son olarak da kelimenin alınabilen en uzun ilk hecesi ve ekleri için üç ayrı statik şablon oluşturulup, Türkçe bir metnin kayıpsız geri dönüşümü sağlanmıştır. Sıkıştırma verimindeki başarıyı daha da arttırmak amacıyla kelimenin gövde-kök ve ekleri için iki ayrı sözlük kullanarak dinamik Huffman kodlaması gerçekleştirilmiştir. Ayrıca kod çözme işleminde ihtiyaç duyulan Huffman ağaç yapısına ait bilginin, sıkıştırılan metnin önüne konan başlık (*header*) alanında tuttuğu yer, bu doktora çalışması kapsamında geliştirilen bir yöntem ile n elemanlı bir Huffman ağacının (2n-2) adet bit ile ifade edilmesi sağlanmış olup, sıkıştırma veriminde %1.5'luk bir artış elde edilmiştir. Veri sıkıştırma tekniklerinin test edilmesinde kullanılan Galgary Corpus ve Catenbury Corpus'a uygun olarak 14 adet Türkçe metinden oluşan bir test kümesi oluşturulmuş ve sistem başarımı bu test kümesi üzerinde incelenerek değerlendirilmiştir.

Türkçe metinler üzerinde yapılan analiz sonucunda, Türkçe bir metin içerisinde geçen kelime kullanım sıklıklarının Zipf kanununa uyum gösterdiği de belirlenmiş ve kullanım sıklıklarına göre matematiksel bir model kurulmuştur.

Anahtar kelimeler : Veri sıkıştırma, Huffman kodlaması, Zipf kanunu, Türkçe'nin biçimbilimsel analizi, Türkçe

ABSTRACT

In this thesis, a new approach for the compression of turkish documents is proposed. In contrast to common data-compression methods, this approach determines the frequency of root, stem,syllables and suffixes in a document through a morphological analysis and uses this output in the generation of dynamic Huffman codes. The evaluation of the proposed approach has been accomplished by implementing the system on several different turkish documents.

The proposed approach distinguishes itself from common place algorithms by the fact that the document to the compressed is evaluated in accordance to the turkish language and rather than as plain binary data hence broken into its roots, stems, suffixes and syllables. Since text compression and decompression requires lossless operation, the proposed method expresses the roots, stems and suffices using Huffman trees to maximize the ratio of coded information to the number of required bits. The header part which carries information to be used during decompression has been optimized by the proposal of a new coding method to describe a Huffman tree which resulted in approximately %1.5 gain in overall compression performance. The proposed coding expresses a Huffman tree with (n) elements in (2n-2) bits and thus reduces the header size clearly. A corpus consisting of 14 turkish documents similar to Galgary Corpus and Catenbury Corpus has been formed and the systems overall performance has been evaluated on this corpus.

Through analysis on turkish documents it has been observed that the frequency distribution of words in a turkish document conforms to the Zipf's law which helps the development of the implemented mathematical model.

Keywords : Data compression, Huffman coding, Zipf law, The morphology of the Turkish Language, Turkish

1. GİRİŞ

Geride bıraktığımız 20. yüzyıl; bilginin elde edilmesi, işlenmesi, dağıtılması ve paylaşılmasının ön plana çıktığı bilişim sektöründe, teknolojinin diğer alanlarından daha çok gelişmenin kaydedildiği bir dönem olmuştur. Bilişim teknolojilerinde yaşanan baş döndürücü gelişmeler, büyük bilgisayar sistemlerinin yanı sıra, kişisel bilgisayarlar alanında da veri kapasitesinin, işlem gücünün ve iletişim hızının katlanarak artmasını ve bu tarz sistemlerin günlük hayatın vazgeçilmez birer parçası olmasını sağlamıştır.

Kurumların tüm bilgi işlem ihtiyacını karşılıyan bir ana bilgisayar sistemi etrafında odaklanmış bilgi işleme modelinden, birbirlerine bilgisayar ağı altyapısı ile bağlı değişik kapasite ve güçte, ama gerektiğinde bağımsız çalışabilen bilgisayarlardan oluşan dağıtık işleme özelliğine sahip sistemlere doğru bir geçiş başlamıştır. Bu geçişle beraber merkezi kaynaklar (dosya sunucu, yazıcı, CPU sunucu vb.) yerine kullanılan altyapı içerisinde dağıtılmış birden fazla kaynağın kullanıcıların hizmetine sunulması yöntemi geçerlilik kazanmıştır.

Bu yaklaşımın doğal sonucu olarak kullanıcılararası ve kullanıcılar ile kullanılan kaynaklar arasındaki bilgi trafiği artmış, en az gecikme ile fiziksel konumundan bağımsız olarak kaynaklara erişme önem kazanmıştır. Bu problemi çözmek amacıyla bir yandan daha geniş kapasiteye sahip bilgisayar sistemleri ile birim zamanda daha fazla bilgi aktarabilecek haberleşme alt yapıları üzerinde çalışılırken, diğer yandan yer ve zamandan kazanmak amacıyla depolanacak veya iletilecek bilginin, uygulamanın özelliğine bağlı olarak kayıplı veya kayıpsız geri dönüşüm sağlanacak şekilde kapladığı yerin azaltılmasına çalışılmaktadır. Bu işlem genel olarak veri sıkıştırma (*data compression*) olarak adlandırılmaktadır.

Kullanıcıların bilgisayar ortamında ürettikleri bilgileri, eskiliğine veya gerekliliğine bakmaksızın, sürekli saklama istekleri, bağlı bulundukları haberleşme ağı içerisinde istedikleri bilgiye birkaç saniyeden fazla bekleyerek erişmeleri durumunda ağın çalışmasını yavaş olarak nitelendirmeleri, özellikle tüm dünyayı kapsayan internet haberleşme ağını da düşünecek olursak, veri sıkıştırmanın önemi daha iyi anlaşılacaktır.

Veri sıkıştırmada kullanılan tekniklerin tarihçesi ve gelişimi incelenecek olursa, ilk tekniklerin, temelde sıkıştırmadan çok kodlama işlemine dayandığı görülür (Braille, 1820; Mors, 1838). İstatistik yöntemlerinin kullanılması ile veri içerisinde geçen bilgilerin sıklığının belirlenmesi ve daha sık kullanılan bilgilerin daha az yer kaplayacak şekilde ifade edilmesi prensibi gündeme gelmiştir.

Shannon-Fano (1949) ve Huffman (1952) kullanım sıklığına bağlı olarak değişen uzunlukta kod kelimesi kullanımı ile veri sıkıştırma konusunda ilk önemli adımı atmışlardır. Veri içerisinde geçen tekrarlamaları azaltan ve dinamik olarak Huffman kodlamasını gerçekleştiren Lempel-Ziv (1977) günümüzün pek çok kayıpsız sıkıştırma yöntemine de temel oluşturmuştur. Bu yöntemler sıkıştırılacak veriyi metin veya ikili bilgi olarak değerlendirmişler, veri içeriğinin üzerinde durmamışlardır. A.S.Fraenkel, M.Mor ve Y.Perl (Fraenkel vd., 1983) İngilizce ve İbranice dillerindeki önek ve soneklerin incelenmesine dayalı sezgisel sıkıştırma yöntemleri önermişlerdir. W.H.Hsu ve A.E.Z.Warico (Hsu vd.,1995), içerisinde metin, ses ve hareketli görüntü (animasyon) gibi farklı yapıda bilgi içeren verilerin üzerinde çalışmışlardır.

Gerçekleştirilen bu doktora çalışmasının konusu "Türkçe metinlerin, Türk dilinin yapısına uygun şekilde analizine ve kelimeye dayalı kayıpsız geri dönüşümü sağlayan dinamik yöntemle sıkıştırılmasına yönelik bir sistem" in geliştirilmesidir. Önerilen sistemin en önemli ve diğer çalışmalara göre ayırt edici özelliği, sıkıştırılacak Türkçe metni biçimbilimsel olarak incelemesi ve bunun sonucunda elde edilen istatistiksel verilere göre kök-gövde, ek ve diğer semboller için dinamik Huffman ağaçlarının oluşturulmasıdır. Bu sayede klasik karaktere bağlı Huffman kodlamasının ötesinde metin içerisindeki kelimeler kök-gövde ve ek ayırımından sonra bir bütün olarak ele alınmışlardır. Yine bu doktora çalışması kapsamında, sıkıştırılan verilerin geri açılmasında kullanılacak Huffman ağaçlarının, başlık (*header*) adı verilen alan içerisinde yer alacak olan sıklık bilgilerinin ifade edilmesi için toplam maliyeti eleman başına 2 biti geçmeyen özel bir kodlama tekniği geliştirilmiştir.

Tez çalışması kapsamında geliştirilen sistemi daha iyi anlatılabilmesi amacıyla ikinci bölümde veri sıkıştırmanın tanımına ve veri sıkıştırma tekniklerinin değerlendirilmesine yönelik kavramlara yer verilmiş, veri sıkıştırma yöntemlerinin sınıflandırılması anlatılmıştır. Üçüncü

bölümde ise kayıpsız geri dönüşümü sağlayan içeriğe dayalı, statik ve dinamik yöntemler alt bölümler halinde tanıtılmıştır. Dördüncü bölümde bugüne kadar ortaya atılan kayıpsız sıkıştırma yöntemlerine ve Türk dili üzerinde yapılan araştırmalara sıra ile değinilmiştir. Beşinci bölümde doktora tezini oluşturan; Türkçe metinlerin, Türk dili kurallarına uygun olarak kök-gövde, ek, hece ve diğer sembollere ayrılması; bu bilgilerin kök-gövde bir bütün, ek,hece ve diğer semboller ise diğer bir bütün olarak değerlendirilerek istatistiksel dağılımlarının çıkartılması; dinamik Huffman kodlama yöntemi ile değişken uzunlukta kod kümeleri ile ifade edilecek şekilde Huffman ağaçlarına yerleştirilmesi; sıkıştırılmış verinin açılması aşamasında kullanılmak üzere Huffman ağaçlarını ve kullanılan kök-gövde, ek ve diğer sembolleri belirten başlık kısmının oluşturulması anlatılmaktadır. Aynı bölümde doktora çalışması kapsamında oluşturulan veri sıkıştırma yöntemi ile sıkıştırılmış verilerin açılması için izlenmesi gereken adımlar ve kullanım sıklıklarının matematiksel modelinin çıkarılması da verilmektedir. Altıncı bölümde ise elde edilen sonuçlar sunulmuş ve bu sonuçların incelemesi yapılmıştır.

2. VERİ SIKIŞTIRMA ile İLGİLİ KAVRAMLAR

2.1 Veri Sıkıştırma Nedir?

Veri sıkıştırma, eldeki verinin yerden ve zamandan tasarruf etmek amacıyla uygulamanın ihtiyacına göre kayıplı (*lossy*) veya kayıpsız (*lossless*) olarak geri dönüşümü sağlayacak bir yöntem ile daha az yer tutacak şekilde işlenmesidir. Veri sıkıştırma, verinin saklanacağı sayısal ortamda daha fazla yer tutmasının fiziksel ve maddi açıdan sorun olmasının yanı sıra iletişim kanallarından birim zamanda aktarılan veri miktarını etkilemesiyle de önemi giderek artan bir konu haline gelmiştir. Bilişim teknolojilerindeki gelişmeler veri sıkıştırma yöntemlerinin yazılım ve donanım elemanlarıyla da gerçekleştirilmesini sağlamıştır. Kullanılan yöntemler, sıkıştırılacak verinin özelliğine bağlı olarak değişik sonuçlar vermektedir. Örneğin, doğal dil metinlerinin sıkıştırılmasında Huffman algoritması en iyi başarıyı sağlarken, bit seviyesinde çok fazla tekrarlamanın olduğu ikili (*binary*) verilerde benzer bir başarıyı elde etmek için RLE (*Run-Length Encoding*) algoritması kullanılmalıdır.

2.2 Veri Sıkıştırmanın Kıstasları

Veri sıkıştırma tekniklerinin yararlarını incelemek için bir karşılaştırma yapısı kurmak gerekir. Burada tartışılan ve ölçülebilecek her olayın iki boyutu vardır. Bunlar algoritma karmaşıklığı ve sıkıştırma oranıdır.

Veri sıkıştırması, bir veri iletişim uygulamasında kullanıldığında amaç hızdır. İletişim hızı, gönderilen bit sayısına, kodlayıcının kodlu mesajı üretmesi için gereken zamana ve kod çözücünün orijinal mesajı yeniden oluşturması için geçen süreye bağlıdır. Bir veri saklama uygulamasında, öncelikli amaç sıkıştırma derecesi olmakla birlikte, algoritmanın pratik gerçeklemesi için algoritma karmaşıklığının az olması gerekir.

2.2.1 Entropy

Bir mesajın ne kadar sıkıştırılabileceği ve sonradan doğru olarak yerine konulabileceğine ilişkin bir sınırın olduğunu ortaya koyan Shannon (1948) bu sınırı "entropy" olarak adlandırdı.

$A=(a_1,a_2,a_3,...,a_n)$ sonlu kümesinden oluşan bir alfabeye sahip bilgi kaynağında, A alfabesinin bazı elemanlarının bir arada kullanımı ile $(a_i a_j a_k \ldots a_r)$ $i,j,k,r \leq n$ şeklinde bir sembol oluşsun. Oluşan bu sembolde $a_1,a_2,a_3,...,a_r$ sembollerinin kullanılma olasılıkları, $P_1 = P(a_1)$, $P_2 = P(a_2)$, $P_3 = P(a_3)...P_r = P(a_r)$ ise, olasılıklar toplamı eşitlik (2.1), entropy'de eşitlik (2.2)'deki gibi olur.

Olasılıklar toplamı : $\sum P_i = 1$ (2.1)

Entropy : $H = -\sum_{i=1}^{n} P_i \log_2 P_i$ (2.2)

- Özel bir durum olarak, olasılıklar birbirine eşit ise $P = P_1 = P_2 = ... = P_n$ olasılıklar toplamı eşitlik (2.3)'deki gibi olur.

$$\sum_{i=1}^{n} P_i = nP = 1 \quad (2.3)$$

- Entropy eşitliği de (2.4) gibi olacaktır.

$$H = -\sum_{i=1}^{n} P_i \log_2 P_i = \sum_{i=1}^{n} P_i \log_2 n \quad (2.4)$$

- Entropy ile ilgili olarak $0 \leq H \leq \log_2 n$ verilebilir.

Eğer $P_1 = 1, \quad P_2 = P_3 = ... = P_n = 0$ ise $H = 0$

Eğer $P_1 = P_2 = P_3 = ... = P_n = 1/n$ ise $H = \log_2 n$

2.2.2 Ortalama mesaj uzunluğu

Huffman (1952) ortalama mesaj uzunluğunu, kaynak alfabesindeki herbir sembolün oluşma olasılığı ile bu sembolü kodlamak için gerekli olan bit sayısının (β_i) çarpımlarının toplamı olarak ifade etmiştir. Ortalama mesaj uzunluğu (2.5)'deki gibi ifade edilir.

$$S = \sum_{i=1}^{n} P_i \beta_i \qquad (2.5)$$

Bu durumda entropy ile ortalama mesaj uzunluğu arasında (2.6)'daki gibi bir bağıntı söz konusudur.

$$H \leq S \leq \lceil \log_2 n \rceil \qquad (2.6)$$

2.2.3 Tekrarlanma (*Redundancy*)

Tekrarlanma, ortalama mesaj uzunluğu ile entropy arasındaki fark olarak ifade edilmiştir (Shannon ve Weaver, 1949). Tekrarlanma eşitlik (2.7)'deki gibi gösterilir.

$$R = -H + S \qquad (2.7)$$

2.2.4 Sıkıştırma oranı (*Compression ratio*)

Sıkıştırma oranı, çıkış bilgisi uzunluğunun, giriş bilgisi uzunluğuna oranı olarak (2.8)'deki gibi tarif edilir.

ρ= Çıkış Bilgisi Uzunluğu / Giriş Bilgisi Uzunluğu (2.8)

Sıkıştırma oranının %60 olarak gerçekleştirildiği bir sistemde, 100 birim uzunluğundaki bir metin, sıkıştırma sonunda 60 birime inmiş demektir. Bir metnin sıkıştırılmış hali aslından büyükse bu işlem "genişletme" adını alır. Bu durumda $\rho > 1$ olur.

2.2.5 Sıkıştırma faktörü (*Compression factor*)

Sıkıştırma faktörü, sıkıştırma oranının tersi olup (2.9)'daki gibi ifade edilir.

$$v = 1/\rho \quad (2.9)$$

2.2.6 Sıkıştırma verimi(η)

Sıkıştırma sonunda elde edilen başarımı, sıkıştırma verimi olarak (2.10)'daki gibi tarif edebiliriz.

$$\eta = 100*(1-\rho) \quad (2.10)$$

2.2.7 Kraft-MacMillan eşitsizliği

Bu eşitsizlik, değişken uzunluklu tek çözümü olan kodlarla ilgilidir. N adet L_i uzunluğundaki değişken uzunluklu kodlar için Kraft-MacMillan eşitsizliği (2.11)'deki gibi yazılır. (2.11)'i sağlayan değişken uzunluklu kod için, tek çözümü olan kod diyebiliriz.

$$\sum_{i=1}^{n} 2^{-L_i} \leq 1 \qquad (2.11)$$

(2.11)'deki eşitsizlik entropy ile ilişkilendirildiğinde L_i kod uzunluğu (2.12)'deki gibi yazılır.

$$L_i = -\log_2 P_i + E_i \qquad (2.12)$$

Burada, E_i, L_i'nin entropy'den büyük kısmı olup,

$2^{-L_i} = 2^{(\log_2 P_i - E_i)} = 2^{\log_2 P_i} / 2^{E_i} = P_i / 2^{E_i}$ olarak yazılabilir.

- Özel bir durum olarak bütün fazla uzunlukların eşit olduğunu kabul edelim. (E_i = E) olarak aldığımızda Kraft eşitsizliği,

$$1 \geq \sum_{i=1}^{n} P_i / 2^E = \left(\sum_{i=1}^{n} P_i \right) / 2^E = 1/2^E \Rightarrow 2^E \geq 1 \Rightarrow E \geq 0$$

olur. Tek çözümü olan kod, negatif olmayan extra uzunluğa sahip, kodun uzunluğu entropy ile tanımlanan uzunluğa eşit veya daha fazla demektir. (2.11)'deki eşitsizliğin kullanımını basit bir örnek ile gösterelim. N tane eşit uzunluklu ikili kod alalım. Her kodun uzunluğu $L_i = \log_2 n$ olsun. Kraft-MacMillan eşitsizliği toplamı,

$$\sum_{i=1}^{n} 2^{-Li} = \sum_{1}^{n} 2^{-\log_2 n} = \sum_{1}^{n} \frac{1}{\log_2 n} = 1$$

olur. Bir diğer örnek; n kodun, birincisi sıkıştırılacak, ikincisi genişletilecek olsun. Bunu $L_1 = \log_2 n - a, L_2 = \log_2 n + e$ ve $L_3 = L_4 = \ldots = L_n = \log_2 n$ şeklinde gösterelim. Burada a ve e pozitif sayılar olup, e>a dır. Bir sembolü a faktörü kadar sıkıştırmak, diğer sembolü bir büyüme faktörü ile genişletmeyi gerektirir. Sıkıştırılan sembolün kullanım sıklığından fazla ise toplam sıkıştırma faktörü SF > 1 olur.

$$\sum_{1}^{n} 2^{-L_i} = 2^{-L_1} + 2^{-L_2} + \sum_{3}^{n} 2^{-\log_2 n}$$

$$= 2^{-\log_2 n + a} + 2^{-\log_2 n - e} + \sum_{1}^{n} 2^{-\log_2 n} - 2 * 2^{-\log_2 n}$$

$$= \frac{2^a}{n} + \frac{2^{-e}}{n} + 1 - \frac{2}{n}$$

Kraft_MacMillan eşitsizliğine göre,

$$= \frac{2^a}{n} + \frac{2^{-e}}{n} + 1 - \frac{2}{n} \le 1; \qquad \frac{2^a}{n} + \frac{2^{-e}}{n} - \frac{2}{n} \le 0 \quad \text{veya} \quad 2^{-e} \le 2 - 2^a$$

$-e \le \log_2(2 - 2^a)$ veya $e \ge \log_2(2 - 2^a)$ olarak ifade edilir. Yukarıdaki eşitsizlikte a ≤ 1 olmalı (aksi halde $2\text{-}2^a$ negatif olur) fakat a değeri (0,1] aralığında bir değere sahip olmalı ve bu aralık e > a şartını sağlamalıdır.

2.3 Kodlara ait Özellikler

Sıkıştırma algoritmaları ile üretilen kodlar dört ana grup altında toplanır.

- **Blok-blok :** Kaynak mesajın ve kod kelimelerinin sabit uzunluklu olduğu durumdur (Çizelge 2.1). Bu gruptaki kodlara *sabit uzunluklu kodlar*'da denir. ASCII* ve EBCDIC** en eski ve en yaygın kullanılan blok-blok kodlardır.

- **Blok-değişken :** Kaynak mesajın sabit, kod kelimenin değişken uzunlukta olduğu durumdur (Çizelge 2.2).

Çizelge 2.1 Blok-blok kod

Kaynak mesaj	Kod kelime
a_1	00
a_2	01
a_3	10
a_4	11

Çizelge 2.2 Blok-değişken kod

Kaynak mesaj	Kod kelime
a_1	0
a_2	10
a_3	110
a_4	111

- **Değişken-Blok :** Kaynak mesajın değişken, kod kelimenin sabit uzunlukta olduğu durumdur (Çizelge 2.3).

- **Değişken-Değişken :** Değişken uzunluklu kaynak mesajın ve değişken uzunluklu kod kelimenin olduğu durumdur (Çizelge 2.4).

Çizelge 2.3 Değişken-blok kod

Kaynak mesaj	Kod kelime
a_1	00
$a_1\ a_2$	01
$a_1\ a_2\ a_3$	10
$a_1\ a_2 a_3\ a_3$	11

Çizelge 2.4 Değişken-değişken kod

Kaynak mesaj	Kod kelime
$a_1\ a_1$	0
$a_2\ a_2\ a_2$	1
$a_3\ a_3\ a_3\ a_3$	10
$a_4\ a_4\ a_4\ a_4\ a_4$	11

* ASCII American National Standard Code for Information
** EBCDIC Extended Binary Coded Decimal Interchange Code

Bunların dışında, sıkıştırma algoritmalarınca üretilen kodların **tek çözümlü** *(uniquely decodable)* ve **anında** (*instantaneous*) çözülebilme gibi farklı özellikleri de vardır.

2.3.1 Değişken uzunluklu (*variable length*) kodlar

$A=(a_1, a_2, a_3, a_4)$ sonlu kümesinden oluşan bir alfabede, a_1, a_2, a_3, a_4 sembollerinin herbiri veri katarında eşit olasılıkla bulunuyorsa,

$P(a_1)= P(a_2)= P(a_3)= P(a_4)=P=0.25$
Entropy, $H = - nP \log_2 P = 2$ olarak hesaplanır.

Bu durumda her sembol 2-bit sabit kod uzunluğu ile kodlanabilir ve tekrar görülmez.

Çizelge 2.5 Sabit ve değişken uzunluklu kodlar

Sembol	Olasılığı	Sabit uzunluklu kod kelime A	Değişken uzunluklu kod kelime B	Değişken uzunluklu kod kelime C
a_1	.49	00	1	1
a_2	.25	01	01	01
a_3	.25	10	010	000
a_4	.01	11	001	001

Her sembol Çizelge 2.3'deki gibi farklı olasılıkla verilmiş olsun. Bu durumda sabit uzunluklu kod kelime A kullanıldığında, entropy ve tekrarlılık,

$$H = -\sum_{i=1}^{n} P_i \log_2 P_i = 1.57 \text{bit} \qquad R = - H + \log_2 4 = 0.43 \text{ bit}$$

olarak hesaplanır. Değişken uzunluklu kod kelime B kullanıldığında ise, ortalama mesaj uzunluğu ve tekrarlılık,

$S = \sum P_i * \beta_i = 1.77$ bit $\qquad R = -H + S = 0.2$ bit

olarak bulunur. Örnekte görüldüğü gibi değişken uzunluklu kodlar kullanıldığında her sembol için fazladan kullanılan bit sayısı daha az olmaktadır.

2.3.1.1 Tek çözümlü (*evrensel*) kodlar

Aynı uzunlukta olmayan kod kelimeleri ile kodlanmış kaynak sembollerinden her seferinde orijinal mesajın elde edilebilmesi gerekir. Çizelge 2.5'de verilen sabit uzunlukluların tamamı tek çözümlü kodlardır.

$a_1\ a_3\ a_2\ a_1\ a_3\ a_3\ a_4\ a_2\ a_1\ a_1$ dizisini değişken uzunluklu kod kelime B ile kodladığımızda, çıkış **1|010|01|1|010|010|001|01|1|1** olur. Bu kodlanmış ifadeden orijinal mesaj dizisi hatasız bir şekilde oluşturulabilinir.

2.3.1.2 Anında (*instantaneous*) çözülebilen kodlar

Çizelge 2.5'de sadece kod kelime B diğer ikisine göre farklılık gösterir. Kod kelime B ile kodlanmış bir binary dizi verildiğinde her kelime alındığı anda çözülemez. Ancak sonraki kelimeler geldiğinde çözülebilir. Bu nedenle kod kelime B'nin anında çözülebilme özelliği yoktur.

2.4 Veri Sıkıştırma Yöntemlerinin Sınıflandırılması

Sıkıştırma yöntemleri genel olarak "**kayıplı**"(*lossy*) ve "**kayıpsız**"(*lossless*) olmak üzere iki ana grupta toplanır. Bilgi depolama ve metin dosyalarının sıkıştırılmasında kayıpsız sıkıştırma şartı aranırken, ses ve görüntü dosyalarının sıkıştırılmasında bütünlüğü bozmayacak kadar kayıplı sıkıştırmaya müsade edilmektedir. Bu doktora çalışmasında kayıpsız sıkıştırma yöntemi esas alınmıştır.

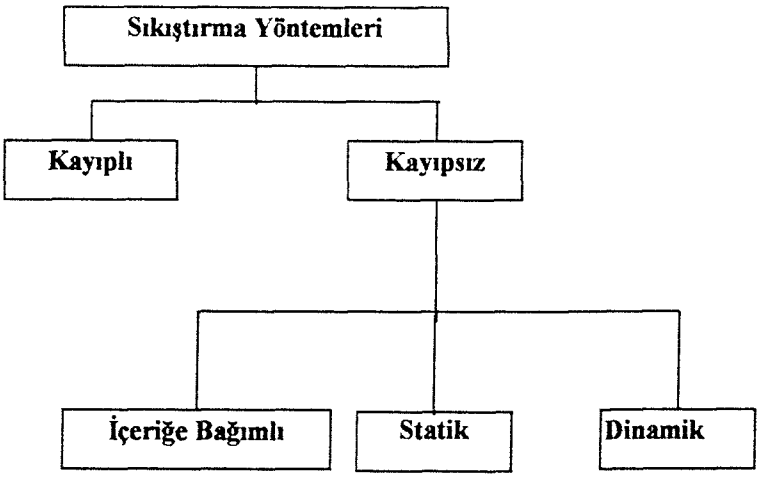
Kayıpsız sıkıştırma yöntemleri de kendi içinde, üç ana gruba ayrılır (Şekil 2.1).

- İçeriğe bağımlı
- Statik
- Dinamik (*adaptif-uyumlu*)

2.4.1 Kayıplı veri sıkıştırma yöntemleri

Verinin yüzde yüz doğru olarak elde edilmesinin zorunlu olmadığı durumlarda kullanılan bu yöntemlerde çok yüksek sıkıştırma oranlarına erişebilmektedir. Sabit ve hareketli görüntüler için bu oran **10:1** ile **100:1** arasında değişebilmektedir.

İnsan gözü ve kulağının algılama kabiliyeti logaritmik değiştiğinden, az kayıplı bir sıkıştırma işleminde görme ve işitme olayındaki bütünlük bozulmasına göz yumulur. Uygulamaya bağlı olarak, uygun seviyedeki bir kayıp son kullanıcı tarafından kabul edilebilir. Değişik kalitede sıkıştırma yöntemleri için değişik algoritmalar vardır. Burada sıkıştırma algoritmasının verimi, kodu çözülen (*decompress*) verinin doğruluğu ile ters orantılıdır.



Şekil 2.1 Sıkıştırma yöntemlerinin sınıflandırması

2.4.2 Kayıpsız veri sıkıştırma yöntemleri

Bu yöntemlerde verinin tekrar elde edilmesini garantilemek üzere, yeterli bilgiyi doğru olarak bulundurmak zorunlu olduğundan, kayıpsız sistemler yüksek sıkıştırma oranlarını yakalayamazlar. Kayıpsız veri sıkıştırma algoritmalarında sıkıştırma oranları **2:1** ile **8:1** değişebilmektedir. Bu oran sıkıştırma algoritmasının yeteneğine ve bilgi kaynağındaki tekrarlama derecesine bağlı olarak değişim gösterir.

2.4.2.1 İçeriğe bağımlı sıkıştırma yöntemleri

Bu yöntem, çeşitli uygulamalarda görülen bazı bölgesel tekrarları azaltmak için kullanılır. Programlandıkları dosyalardaki fazlalıklara çok iyi adapte oldukları için bu tip dosyalarda genel sıkıştırma yöntemlerinden daha fazla başarı alınır. En büyük dezavantajı, sınırlı sayıda uygulamada kullanılabilmesidir. Resim dosyalarında RLE yaygın olarak kullanılan bir içeriğe bağımlı sıkıştırma yöntemidir.

Metin dosyalarına uygulanabilen bir çok yöntem vardır. Bunlardan biri, sözlük kullanarak metinde geçen bütün ya da yaygın olarak kullanılan bazı sözcükleri, sözlüğe bağlayarak şifrelemektir. Diğer bir uygulama da kullanılan deyimler ve karakter çiftlerinin şifrelenmesidir.

2.4.2.2 Statik sıkıştırma yöntemleri

Statik sıkıştırma yöntemlerinde önceden yapılan bir araştırma ile kullanım sıklıkları bulunur. Sıkıştırma algoritması tasarlanırken daha önceden belirlenen kullanım sıklıkları için tekrarı azaltmak üzere uygun kodlar oluşturulur.

1952'de ortaya atılan Huffman Kodlama tekniği (Knuth, 1985) günümüzde de yaygın olarak kullanılan bir yöntemdir. Bu yöntem 1949'da çıkan Shannon-Fano sıkıştırma yöntemine benzemekle birlikte Huffman Metodu daha başarılı kodlar üretmektedir. Shannon-Fano yöntemi, sembollerin olasılıkları ikinin negatif kuvvetleri şeklinde ise, daha iyi sonuç vermektedir. İki yöntem arasındaki temel fark: Shannon-Fano (Salomon, 1998), kodları soldan sağa doğru oluştururken, Huffman, kodları sağdan sola doğru oluşturur.

Aritmetik kodlama ise veri sıkıştırmaya diğer statik yöntemlerden farklı şekilde yaklaşım göstermiştir. Bu yöntem kaynak mesajların kod kelimelere eşlendiği bir kod yaratmaz. Aritmetik kodlama kaynağın entropy'sine oldukça yakın sonuçlar elde eder.

2.4.2.3 Dinamik sıkıştırma yöntemleri

Dinamik sıkıştırma yöntemlerinde, veride geçen kullanım sıklığının önceden bilinmesine gerek yoktur. Kodlama anında veride bulunan tekrara göre kendilerini adapte ederler. Dinamik algoritmalar, statik sıkıştırma yöntemlerinin dinamik uygulaması ve sözlük kullanan yöntemler olarak kendi içerisinde ikiye ayrılırlar. Statik yöntemlerden Huffman, Shanon-Fano ve Aritmetik kodlamanın dinamik uygulamaları gerçekleştirilmiştir. Sözlük kullanan yöntemlere ise Lempel ve Ziv algoritmaları ve bu algoritmaların türevleri örnek olarak gösterilir.

2.5 Doğal Dillerin Analizinde Zipf Kanunu

Bir metin içerisinde geçen değişik kelime sayısı t ise ve bunların herbiri kullanım sıklıkları sırasına göre $n_1,n_2,n_3,...,n_t$ kere kullanılmış ise, metindeki toplam kelime sayısı n olup, $n_1 + n_2 + n_3 + ... + n_t = n$ toplamları, (2.13)'deki gibi yazılır.

$$\sum_{r=1}^{t} n_r = n \qquad (2.13)$$

Amerikalı sosyolog George Kingsley Zipf tarafından ortaya atılan teoriye göre incelenen metindeki n adet kelime içerisinde en sık kullanılan kelime n_1 ise, ikinci sıklıkta kullanılan n_2 ve daha sonra $n_3, n_4, \ldots$ diye değişiyorsa ise, kelimelerin kullanımlarındaki kademelerde $n_1, n_2, n_3, \ldots$ için $1,2,3,\ldots,r$ (*rank*) diye alınırsa, (2.14)'deki denklem elde edilir. Buradan,

$$f_1 = \frac{n_1}{n}, f_2 = \frac{n_2}{n}, \ldots, f_r = \frac{n_r}{n} \qquad (2.14)$$

$$\sum_{r=1}^{t} n_r = n$$

$$\sum_{r=1}^{t} f_r . n = n \sum_{r=1}^{t} f_r \Rightarrow \sum_{r=1}^{t} f_r = 1$$

yazılabilir. Burada f frekansları, her kelime türünün n elemanlı metindeki kullanım ihtimallerini verir. Zipf'in ortaya attığı teoriye göre,

$$f_r . r^{\alpha} = c = \text{sabit} \qquad (2.15)$$

olup, $f_1 . 1^{\alpha} = f_2 . 2^{\alpha} = f_3 . 3^{\alpha} = \ldots = f_r . r^{\alpha} = c$ dir.

Bu durumda (α=1) özel durumu için,

$$f_r . r = c \qquad (2.16)$$

$$f_1 = f_2 2 = f_3 3 = \ldots = f_r r = c$$

$$f_2 = \frac{f_1}{2}, f_3 = \frac{f_1}{3}, \ldots, f_r = \frac{f_1}{r}$$

olarak yazılabilir. $n_r = f_r.n$ ifadesinde, f_r yerine (2.16)'daki eşitliği koyarsak, (2.17) elde edilmiş olur.

$$n_r = \frac{c}{r} n \qquad (2.17)$$

$\sum_{r=1}^{t} P_r = \sum_{r=1}^{t} f_r = \sum_{r=1}^{t} \frac{n_r}{n} = \frac{1}{n} \sum_{r=1}^{t} n_r = \frac{1}{n} n = 1$ olup, olasılıklar toplamının bire eşit olduğu gösterilir. Toplam n_r ifadesini olasılıkları cinsinden yazacak olursak,

$$\sum_{r=1}^{t} n_r = \sum_{r=1}^{t} f_r n = n \sum_{r=1}^{t} \frac{c}{r} = nc \sum_{r=1}^{t} \frac{1}{r}$$

$\sum_{r=1}^{t} \frac{1}{r}$ toplam dizisi, $H = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{t}$ şeklinde yazılan bir harmonik seri olup,

$\sum_{r=1}^{t} \frac{1}{r}$ toplama işlemi, sürekli değişimle bir integrale dönüştürülürse,

$T = \int_{1}^{t} \frac{1}{r} dr = \ln r \Big|_{1}^{t} = \ln(t) - \ln(1) = \ln(t)$ olacaktır. T'nin bu değeri ile harmonik seri toplamının (H) değeri arasındaki bağıntı, $\sum_{r=1}^{t} \frac{1}{r} \cong \ln(t) + \gamma + \frac{1}{2t} - \frac{1}{12t(t-1)}$....şeklinde yazılır. Euler sabiti değeri $\gamma \cong 0.577218$ olup, c sabiti (2.18)'deki şeklini alır.

$$\sum_{r=1}^{t} n_r = nc[\ln(t) + 0.5772] = n$$

$$\Rightarrow c[\ln(t) + 0.5772] = 1$$

$$c \cong \frac{1}{\ln(t) + 0.5772} \tag{2.18}$$

Eşitlik(2.17)'den yararlanarak t ve c'ye bağlı yeni bir eşitlik elde etmeye çalışalım.

$$c = \frac{r}{n} n_r \tag{2.19}$$

Eşitlik (2.19)'da $r_{max}=t$ $n_{rmax}=1$ olarak alındığında,(2.20) elde edilir.

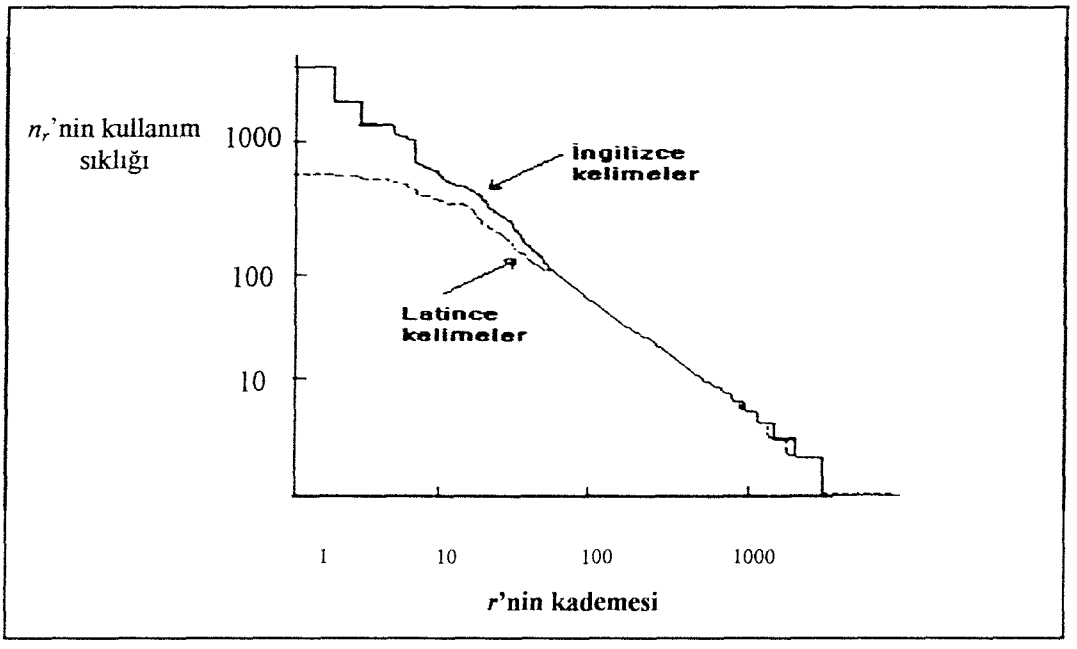
$$c = \frac{t}{n} \tag{2.20}$$

Eşitlik (2.18), eşitlik (2.20)'de yerine konulursa (2.21)'deki eşitlik elde edilir.

$$n = t(\ln(t) + 0.5772) \tag{2.21}$$

100 farklı kelime içeren bir makalede toplam kaç kelime olduğunu yaklaşık n=100(ln(100)+0.5772)=518 olarak söyleyebiliriz.

Zipf kanunun İngilizce'ye ve Latince'ye uygulanması ile elde edilen değerler yardımıyla logaritmik eksenlere çizilen değişim eğrisi (Zipf, 1965) Şekil 2.2'de verilmiştir.



Şekil 2.2 Zipf kanunun İngilizce ve Latince dillerine uygulanması

3. VERİ SIKIŞTIRMA YÖNTEMLERİ

3.1 İçeriğe Dayalı Yöntemler

İçeriğe dayalı yöntemler, çeşitli uygulamalarda bir metin içinde yer alan bölgesel tekrarları kaldırmak için kullanılır. Bu tekrarlar arka arkaya gelen boşluklar, karakter veya semboller olabilir. Bu yöntemin dezavantajı, sadece programlandıkları veri dosyaları için mükemmel sonuç vermeleridir. Bu yöntemdeki metotlar **Örüntü Yerdeğiştirme** (*Patern subsition*), **Ardışıl Karakter Sıkıştırma** (*Run-Length Encoding, RLE*), **Boşluk Sıkıştırma** (*NULL Suppression*), **Bit Adresleme**, **Yarım Sekizli Paketleme** (*Half-byte Packing*), **Bağıl Kodlama**'yı *(Relative Encoding)* örnek olarak verebiliriz.

3.1.1 Örüntü yerdeğiştirme

Örüntü yerdeğiştirme tekniği (*Patern subsition*) veri içerisinde çok sık kullanılan bit örüntüleri, sözcükler, heceler yerine daha kısa kodlar veya semboller koyma tekniğine dayanır. Bir resim verisinin sıkıştırılmasında kullanılan bu teknikte bir satırda x(1),x(2),x(3),...,x(n) görüntü bilgileri olsun ve bu görüntü bilgileri satır satır işlenip (c,l) çiftine yerleştirilsin (c: parlaklık/renk, l:tekrar sayısı). Yani sıkıştırma anında arka arkaya gelen semboller renk ve sıklık bilgisi çifti olarak eklenirler (Gonzalez ve Wintz 1977). Bu yöntem ile ilgili Wilkins (1971) ve Cappellini'nin (1985) çalışmaları bulunmaktadır. Bu yöntemi kullanarak herhangi bir dilde yazılmış olan metin dosyalarının sıkıştırılmasında da başarı sağlanır. Örneğin, Türkçe'de en çok kullanılan sözcük, hece veya eklere kısa kodlar vererek sıkıştırma yapabiliriz.

3.1.2 Ardışıl karakter sıkıştırma

Ardışıl karakter sıkıştırma (*Run-Length Encoding, RLE*), arka arkaya tekrar eden karakterleri sıkıştıran bir yöntemdir. Kodlamada, tekrarın olmadığı karakterler olduğu gibi kodlanırken, tekrarı olan karakterler Şekil 3.1'deki format ile kodlanırlar. Bu yöntemde arka arkaya tekrar eden karakterin kullanım sıklığı 3'den fazla olursa başarı sağlanır. Şekil 3.1'deki formatta ilk byte sıkıştırmanın olduğunu, S_b, ikinci byte tekrar eden karakteri, T_k, üçüncü byte da tekrar sayısını, T_s göstermektedir.

S_b	T_k	T_s

Şekil 3.1 RLE sıkıştırma formatı

Tekrar sayısı* genelde 8 bitlik olup bazı uygulamalarda 16 bit'dir. N dosyadaki karakter sayısı, M tekrar sayısı ve L'de M kere tekrar eden katarın ortalama uzunluğu ise, RLE'de sıkıştırma oranı (Salomon 1998) (3.1)'deki gibi hesaplanır.

$$\rho = \frac{N}{N - M(L-3)} \qquad (3.1)$$

3.1.3 Boşluk sıkıştırma

Boşluk sıkıştırma (*NULL supression*) yöntemi, arka arkaya gelen boşlukları sıkıştırırken, karakterleri ve arka arkaya tekrar etmeyen boşlukları sıkıştırmaz (Ruth ve Kreutzer, 1972; Held 1988). Sıkıştırılacak olan boşluk katarı Şekil 3.2'de gösterilen format ile yer değiştirir. S_i, sıkıştırma işaretcisi, T_s'de tekrar sayısıdır.

*8 bit için RLE-8, 16 bit için RLE-16 kullanılır.

S_i	T_s

Şekil 3.2 Boşluk sıkıştırma formatı

Bir adet sıkıştırma formatı ile 255 tane boşluğu iki byte ile ifade edebiliriz. Bu yöntemde 3 adetten fazla boşluğun ard arda tekrar edilmesi ile başarı artar.

3.1.4 Bit adresleme

Bit adresleme yöntemi ile boşluk sıkıştırmanın olumsuz bir yönü ortadan kaldırılmıştır. Boşluk sıkıştırmada, boşluk karakterlerinin arka arkaya gelmesi gerekiyor ve boşluk karakterleri arasında başka bir karakter varsa sıkıştırma yapılamıyordu. Bit adresleme yöntemi ile bu durum engellenmiştir (Çölkesen, 1995). Kodlanacak verimiz **A□□B□C□□** olarak verilsin. A_b 'de adres bilgisi olup, kodlanacak veride, □ yerine 0, diğer karakterler yerine de 1 yazıldığında 137 değerini elde ederiz. Kodlama işlemi gerçekleştirilirken Şekil 3.3'deki format kullanılmıştır.

A_b	A	B	C

Şekil 3.3 Bit adresleme formatı

3.1.5 Yarım sekizli paketleme

Yarım sekizli paketleme'de (*Half-byte packing*), sıkıştırılacak veri içerisinde bulunan karakterlerin birçoğunun yüksek anlamlı dört bitindeki değer aynı olabilir (EBCDIC formatında sayısal karakterlerin yüksek anlamlı dört biti aynıdır). Bu durumda sıkıştırma formatı Şekil 3.4'de olduğu gibi kodlanır.

Pk_s	S_1	S_2	.....	S_i

Şekil 3.4 Yarım-sekizli paketleme sıkıştırma formatı

P_{ks}, paketlenen kod sayısını, S_i ise karakterin alçak anlamlı dört bitini gösterir.

3.1.6 Bağıl kodlama

Bağıl kodlama (*Relative Encoding veya Differencing*) yönteminde, ard arda gelen verilerin içerikleri birbirlerine yakın olabilir. Bu durumda verilerin kendisini saklamak yerine aralarındaki farkı alarak kodlama işlemini gerçekleştirebiliriz (Gottlieb, 1975). Farklar 0 ile ±255 arasında olmalıdır. Farklar bu değerden büyük ise veri olduğu gibi kodlanır.

Örnek:

Veri : 70, 71, 72.5, 73.1 ise

Kodlanmış veri : 70, 1, 1.5, 0.6 olacaktır.

3.2 Statik KodlamaYöntemleri

Statik sıkıştırma yöntemlerinde önceden yapılan bir araştırma ile kullanım sıklıkları bulunur. Sıkıştırma algoritması tasarlanırken daha önceden belirlenen kullanım sıklıkları için tekrarı azaltmak üzere uygun kodlar oluşturulur.

3.2.1 Shannon-Fano kodlaması

Bu yöntem değişken uzunluklu verileri kodlamak için geliştirilmiş ilk algoritmadır. N sayıda sembolden oluşan ve sembollerinin olasılıkları büyükten küçüğe doğru sıralı şekilde

yerleştirilen bir sembol kümesi verilmiş olsun (Çizelge 3.1). Sembol kümesi, her iki bölümünde kalan olasılıklar toplamı eşit veya eşite yakın olacak şekilde ikiye bölünür. Üst tarafta kalan kümenin sembolleri 0, alt tarafta kalan kümenin sembolleri de 1 ile kodlanır. Her küme tekrar ikiye bölünür ve kodun ikinci biti oluşturulur. Bu işlem kümede iki eleman kalana kadar tekrarlanır. İki elemanı kalan kümenin kodlarına bir bit daha ilave edilir (Çizelge 3.2).

Çizelge 3.1 Shannon-Fano kodlaması (Özel durum)

Sembol kümesi	Olasılık	Kodkelime
a_1	1/2	0
a_2	1/4	10
a_3	1/8	110
a_4	1/16	1110
a_5	1/32	11110
a_6	1/64	111110
a_7	1/64	111111

Çizelge 3.1' de verilen uygulamada entropy, H = 1.96 bit, ortalama mesaj uzunluğu, S = H = 1.96 bit ve tekrarlanma, R = 0 olarak hesaplanır. Shannon-Fano yöntemi, sembollere ait olasılıkların, her sembolün olasılığının bir öncekinin yarısı değerde olması halinde $\left[P_i = \frac{1}{2^i} \Rightarrow P_i = 2^{-i} \quad (i = 1,2,3...)\right]$ maksimum sıkıştırma elde eder. Sıkıştırma oranı, $\rho = 1.96/8 = 0.24$ dür. Bunun nedeni verilen küme iki alt kümeye bölündüğünde, her iki alt kümede kalan sembollerin olasılıkları toplamı birbirine eşit olmakta ve dengeli bir bölünme gerçekleşmektedir.

Uygulamada sembol kümesi elemanlarının Çizelge 3.1'deki gibi ikinin negatif kuvvetleri şeklinde olması ender görülen bir durumdur. Bunun dışındaki durumlarda yöntemin başarısını incelemek üzere Çizelge 3.2'de verilen değerlere göre ortalama mesaj uzunluğunu hesaplayacak olursak, ortalama mesaj uzunluğu, **S=2.95 bit**' dir.

Çizelge 3.2 Shannon-Fano kodlaması

Sembol kümesi	Olasılık	Adımlar	Kod kelime
a_1	8/40	0 0	0 0
a_2	7/40	0 1 0	0 1 0
a_3	6/40	0 1 1	0 1 1
a_4	5/40	1 0 0	1 0 0
a_5	5/40	1 0 1	1 0 1
a_6	4/40	1 1 0	1 1 0
a_7	3/40	1 1 1 0	1 1 1 0
a_8	2/40	1 1 1 1	1 1 1 1

Bu uygulamada entropy, **H = 2.89 bit**'dir ve kodlama ile elde edilen ortalama mesaj uzunluğu bu sonuca oldukça yakındır. Tekrar bit sayısı, **R = 0.035 bit**'dir. Bu uygulama için Shannon-Fano kodlama yönteminin ürettiği mesaj uzunluğu aynı uygulama için Huffman 'nın ürettiği ortalama mesaj uzunluğu ile aynıdır.

3.2.2 Statik Huffman kodlaması

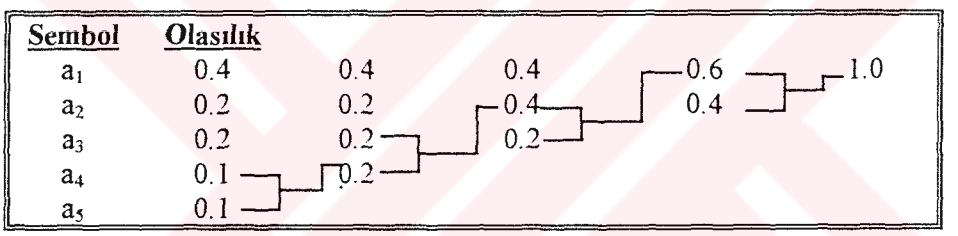
Huffman, veri sıkıştırmada sıklıkla kullanılan bir yöntemdir. Huffman yöntemi bir dereceye kadar Shannon-Fano'ya benzer bir yöntem olmakla beraber daha iyi kodlar üretmektedir.

Huffman algoritması grafik olarak, negatif olmayan $\{w_1,w_2,...,w_n\}$ sıralı ağırlıklar listesini giriş olarak alır ve yaprakları (*leave*) ağırlıklar olan tam bir ikili ağaç (*binary tree*) oluşturur. Tam ikili ağaç, her düğümü iki yaprağı olan ya da hiç yaprağı olmayan ağaçtır. Kod oluşturmak için Huffman algoritması kullanıldığında küçükten büyüğe doğru sıralı ağırlıklar (*kullanım sıklıkları*) listesinin ilk iki elemanı (w_i,w_j), Huffman ağacının ilk iki yaprağını oluştururken, bu iki değer ana listeden çıkarılır. Buna karşı, bu iki elemanın toplamı olan (w_i+w_j) değeri listeye sıralı düzeni bozmayacak şekilde yerleştirilir. Böylece algoritmanın her aşamasında listeden bir eleman eksilir ve n elemanlı semboller listesinde (n-1) adım tekrarlanma sonunda listede tek eleman kalır. Bu eleman, kullanım sıklıklarının toplamı değerinde olup Huffman ağacının kökünü oluşturur. En küçük ağırlık çiftini seçerken birden fazla yol varsa, bu çiftlerden herhangi biri tercih edilir. Bu tercih kod kelimelerinin farklı çıkmasına neden olur. Ancak ortalama mesaj uzunluğunu değiştirmez.

Huffman algoritması, sembol kümesinin her sembolüne karşı gelen kod kelime uzunluklarını belirler. Kod kelimelerinin seçiminde bir çok yol vardır. Ancak kodların önekli olması gerekmektedir. Normal belirlemede her düğümün soldaki dalına "0", sağdakine de "1" rakamı verilir. Her sembol için kod kelimesi, kökten o sembolü temsil eden yaprağa giden yoldaki 0 ve 1 sembollerinden oluşur.

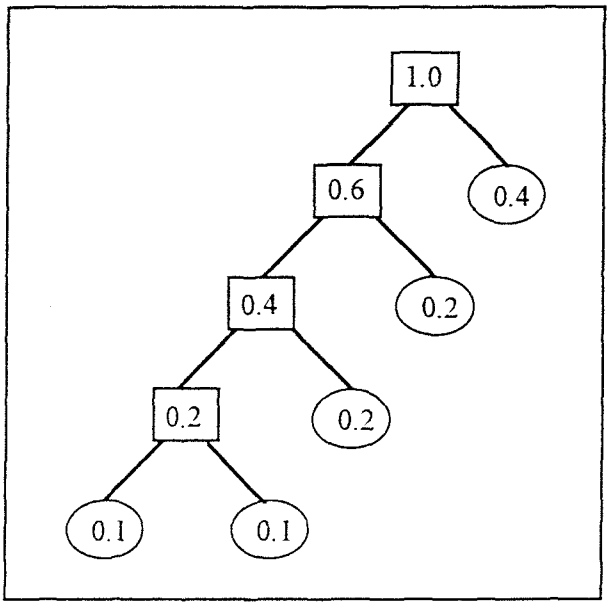
Şekil 3.5'de beş sembolden oluşan, { a_1, a_2, a_3, a_4, a_5 }, bir mesajın sembollerinin olasılıkları gösterilmektedir. Bu olasılıklara bağlı olarak Huffman Liste yapısı oluşturulmuş ve sembol kümesi için Huffman ikili ağacı Şekil 3.6'da gösterilmektedir.

Şekil 3.5'deki olasılıklar için entropy, **H = 2.12 bit** olup, Huffman kod kelimeleri de Çizelge 3.3 'de gösterilmektedir. Üretilen bu kod kelimeleri için ortalama mesaj uzunluğu, **S = 2.2bit** dir.



Şekil 3.5 Huffman kodlama - liste yapısı

Kod kelime oluşturmada birden fazla yol olduğu için Huffman şifreleme her zaman aynı uzunlukta kod kelime elde etmeyebilir. Schwartz (Schwartz,1964) yeni oluşturulan ağırlığı her zaman listede aynı ağırlıkta olanların üzerine yerleştiren ve listenin en altındaki iki ağırlığı birleştiren bir Huffman varyasyonu önermiştir. Böylelikle bulunan kodlar maksimum kod kelime uzunluğunu (Max$\{l_i\}$) ve kod kelime uzunlukları toplamını ($\sum l_i$) azaltır.

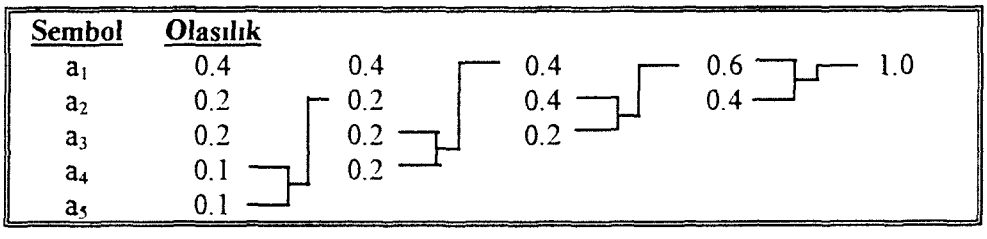


Şekil 3.6 Huffman kodlama-ikili ağaç yapısı

Çizelge 3.3 Verilen olasılık tablosu için Huffman kod kelimeleri

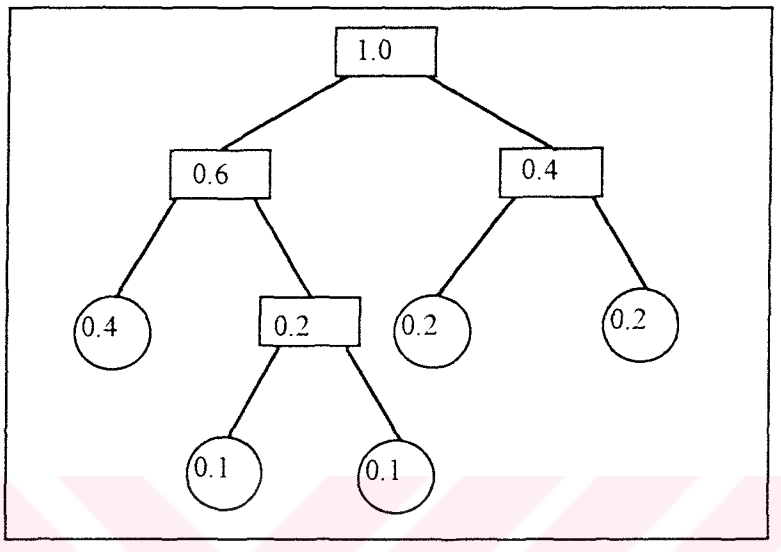
Sembol	Kod kelime
a_1	1
a_2	01
a_3	001
a_4	0000
a_5	0001

Schwartz'ın önerdiği Huffman varyasyonuna göre Şekil 3.5'deki sembol kümesi olasılık değerlerine göre tekrar kod kelime oluşturulursa liste yapısı Şekil 3.7'de, Schwartz Huffman Kodlama ikili ağaç yapısı da Şekil 3.8'de gösterildiği gibi olur.



Şekil 3.7 Schwartz Huffman kodlama-liste yapısı

Şekil 3.7'deki liste yapısı için oluşturulan Schwartz Huffman kod kelimeleri Çizelge 3.4'de gösterilmektedir. Üretilen bu kod kelimeleri için ortalama mesaj uzunluğu, **S = 2.2 bit**'dir. Bu aşamada üretilen bu kodlardan hangisinin daha iyi olduğuna karar verilmesi gerekir. Bunun için üretilen kodlardan daha düşük varyanslı olan tercih edilmelidir.



Şekil 3.8 Schwartz Huffman kodlama ikili ağaç yapısı

Çizelge 3.4 Verilen olasılık tablosu için Schwartz Huffman kod kelimeleri

Sembol	Kodkelime
a_1	00
a_2	10
a_3	11
a_4	010
a_5	011

$$\sigma^2 = \sum_{i=1}^{n} P_i (l_i - S)^2 \qquad (3.2)$$

(3.2) ifadesinden her iki yöntemle üretilen kodlar için varyanslar bulunur. Huffman algoritması için varyans σ^2=1.36 iken, Schwartz Huffman kodlaması için varyans σ^2=0.16 olur. Bu durumda varyansı düşük olan kodlama yöntemi tercih edilir.

3.2.3 Aritmetik kodlama

Aritmetik kodlamada (Witten vd., 1987), sembollerin herbirine bir kod atamak yerine sembol dizisinin tamamına bir kod atanır. Bu yöntemde sembol dizisi teker teker okunur ve okunan her giriş sembolü için oluşturulan koda daha fazla bit eklenir. Bu yöntemi daha iyi anlayabilmek için, sonuçta oluşan kodun [0,1) aralığında kesirli bir sayı olduğu düşünülmelidir. Bu nedenle "9746509" kodu "0". kısmı çıkış dizisine dahil edilmeyeceği halde 0.9746509 olarak algılanmalıdır.

İlk aşamada her sembolün kullanım sıklıkları hesaplanmalı veya tahmin edilmelidir (Çizelge 3.5). İyi bir sonuç için sembol dizisinin tamamının ilk geçişte okunması ve her sembolün kullanım sıklığının tam olarak hesaplanması gereklidir. İkinci geçişte sıkıştırma işlemi yapılmalıdır. Eğer bir program yardımı ile farklı bir kaynaktan kullanım sıklıkları başarılı bir şekilde tahmin edilirse, ilk geçiş işlemi atlanabilinir.

a_1,a_2,a_3 sembolleri, $P_1 = 0.4$, $P_2 = 0.5$ ve $P_3 = 0.1$ olasılıkları ile veriliyor olsun. [0,1) aralığı, her sembolün olasılığı ile orantılı olarak 3 alt aralığa ayrılır (Çizelge 3.5).

Çizelge 3.5 Üç sembol için kullanım sıklıkları ve aralıkları

Semboller	Olasılıklar	Aralıklar
a_1	P_1=0.4	[0,0.4)
a_2	P_2=0.5	[0.4,0.9)
a_3	P_3=0.1	[0.9,1)

Verilen kaynak sembollerine göre "$a_2a_2a_2a_3$" mesajını kodlamak için mevcut aralık [0,1) iken;

YeniÜst = EskiAlt + Aralık * ÜstSınır(x);
YeniAlt = EskiAlt + Aralık * AltSınır(x);
ifadesinden faydalanılarak her **x** sembolü için mevcut aralık tekrar hesaplanır.

- Mesajın ilk sembolü a_2, mevcut aralığı yukarıdaki dönüşüm ifadesinden faydalanarak [0.4,0.9) alt aralığına düşürür.
- Mesajın ikinci sembolü a_2, mevcut aralığı [0.6 , 0.85)

YeniÜst = 0.4 + 0.5 * 0.9 = 0.85

YeniAlt = 0.4 + 0.5 * 0.4 = 0.6 yapar.

- Mesajın üçüncü sembolü a_2, mevcut aralığı [0.7 , 0.825)

YeniÜst = 0.6 + 0.25 * 0.9 = 0.825

YeniAlt = 0.6 + 0.25 * 0.4 = 0.7 yapar.

- Mesajın dördüncü sembolü a_3, mevcut aralığı [0.8125 , 0.825)

YeniÜst = 0.7 + 0.125 * 1 = 0.825

YeniAlt = 0.7 + 0.125 *0.9 = 0.8125 yapar.

Sonuçta karşı tarafa yollanacak olan kod mevcut aralığın alt değeridir (0.8125). Bu değerler Çizelge 3.6'da verilmiştir.

Çizelge 3.6 Verilen giriş dizisi için aritmetik kodlama işlemi

Mesaj	Yeni aralık hesabı	Mevcut aralık
a_2	YeniÜst=0.9 YeniAlt=0.4	[0.4 , 0.9)
a_2	YeniÜst=0.85 YeniAlt=0.6	[0.6 , 0.85)
a_2	YeniÜst=0.825 YeniAlt=0.7	[0.7 , 0.825)
a_3	YeniÜst=0.825 YeniAlt=0.8125	[0.825 , 0.8125)

Aritmetik kodlamada temel adımları özetlemek için aşağıdaki kurallar verilebilir.

1- "Mevcut aralığı" (*current interval*), [0,1) olarak tanımlayarak işleme başlanır.

2- Giriş dizisindeki her **x** sembolü için aşağıdaki iki adım tekrarlanır;

- Her **x** sembolünün olasılığı ile orantılı olarak mevcut aralığı alt aralıklara bölmek.
- Sembol dizisindeki her **x** sembolü için alt aralığı oluşturup, bunu mevcut aralık olarak tanımlamak.

3- Sembol dizisinin tamamı bu şekilde işlendikten sonra, mevcut aralığın alt sınırı çıkış için evrensel bir koddur.

İşlenen her sembol için mevcut aralık daha da küçülür ve bu aralığı göstermek için gerekli bit sayısı artar. Sonuçtaki çıkış tek bir reel sayıdır ve her sembol için bir kod içermez. Ortalama mesaj uzunluğu, bit bazında, çıkış uzunluğunun giriş dizisindeki sembollerin uzunluğuna oranıdır.

Kod çözme işlemi bunun tersi şekilde çalışır. Gelen kodun, ilk dijiti 8 olduğu (0.8125) için kod çözücü bunun [0.4 , 0.9) alt aralığında kalan a_2 olduğunu bulur. Mesajın ikinci sembolü için kod (3.3) yardımı ile tekrar hesaplanır.

$$\text{Kod} = (\text{Kod} - \text{AltSınır}(x)) / \text{Aralık} \qquad (3.3)$$

yeni hesaplanan kod ve kod çözme işlemi Çizelge 3.7'de gösterilmektedir. Kod çözme işlemi sonunda mesaj "$a_2\ a_2\ a_2\ a_3$" geri elde edilmiş olur.

Çizelge 3.7 Aritmetik kodlamada kod çözme işlemi

Mevcut kod	**Sembol**	**Yeni kod**
0.8125	a_2	(0.8125-0.4)/0.5=0.825
0.825	a_2	(0.825-0.4)/0.5=0.85
0.85	a_2	(0.85-0.4)/0.5=0.9
0.9	a_3	(0.9-0.9)/0.1=0

3.3 Dinamik Kodlama Yöntemleri

Dinamik sıkıştırma yöntemlerinde, veride geçen kullanım sıklığının önceden bilinmesine gerek yoktur. Kodlama anında veride bulunan tekrara göre kendilerini adapte ederler. Dinamik algoritmalar, statik sıkıştırma yöntemlerinin dinamik uygulaması ve sözlük kullanan yöntemler olarak kendi içerisinde ikiye ayrılırlar.

3.3.1 Sıklığa bağlı dinamik yöntemler

3.3.1.1 Dinamik Huffman kodlaması

Huffman kodlamasında, alfabedeki bütün sembollerin kullanım sıklıklarının sıkıştırıcı tarafından bilindiği kabul edilir. Bunun için önerilen birinci çözümde sıkıştırıcı orijinal veriyi iki kez okur. Birinci okumada kullanım sıklıkları hesaplanır. Bulunan kullanım sıklıkları kullanılarak yapılan ikinci okumada okunan veri sıkıştırılır. İki okuma arasında, sıkıştırıcı Huffman ağacını oluşturur. Böyle bir yöntem yarı adaptif (*semi-adaptif*) olarak isimlendirilir.

Pratikte adaptif/dinamik Huffman kodlama yöntemi kullanılır. Bu yöntem "UNIX compact" programınında temelini oluşturur. Dinamik Huffman, Faller ve Gallager (Faller,1973; Gallager,1978) tarafından birbirinden bağımsız çalışmalarla geliştirilmiş ve daha sonra Knuth'da (Knuth, 1985) algoritmaya bazı iyileştirmeler ilave etmiş ve sonuçta bu yöntem FGK algoritması olarak anılmaya başlamıştır.

Sıkıştıran ve sıkıştırılmış dosyayı açan algoritmalar için ana fikir boş bir Huffman ağacı ile başlamaktır. Daha sonra okunan ve sıkıştırılan her sembol için bu ilk ağaç değiştirilir. Sıkıştırma ve kod çözme algoritmalarının bir adımdan diğerine geçişte (kodlar değişse bile

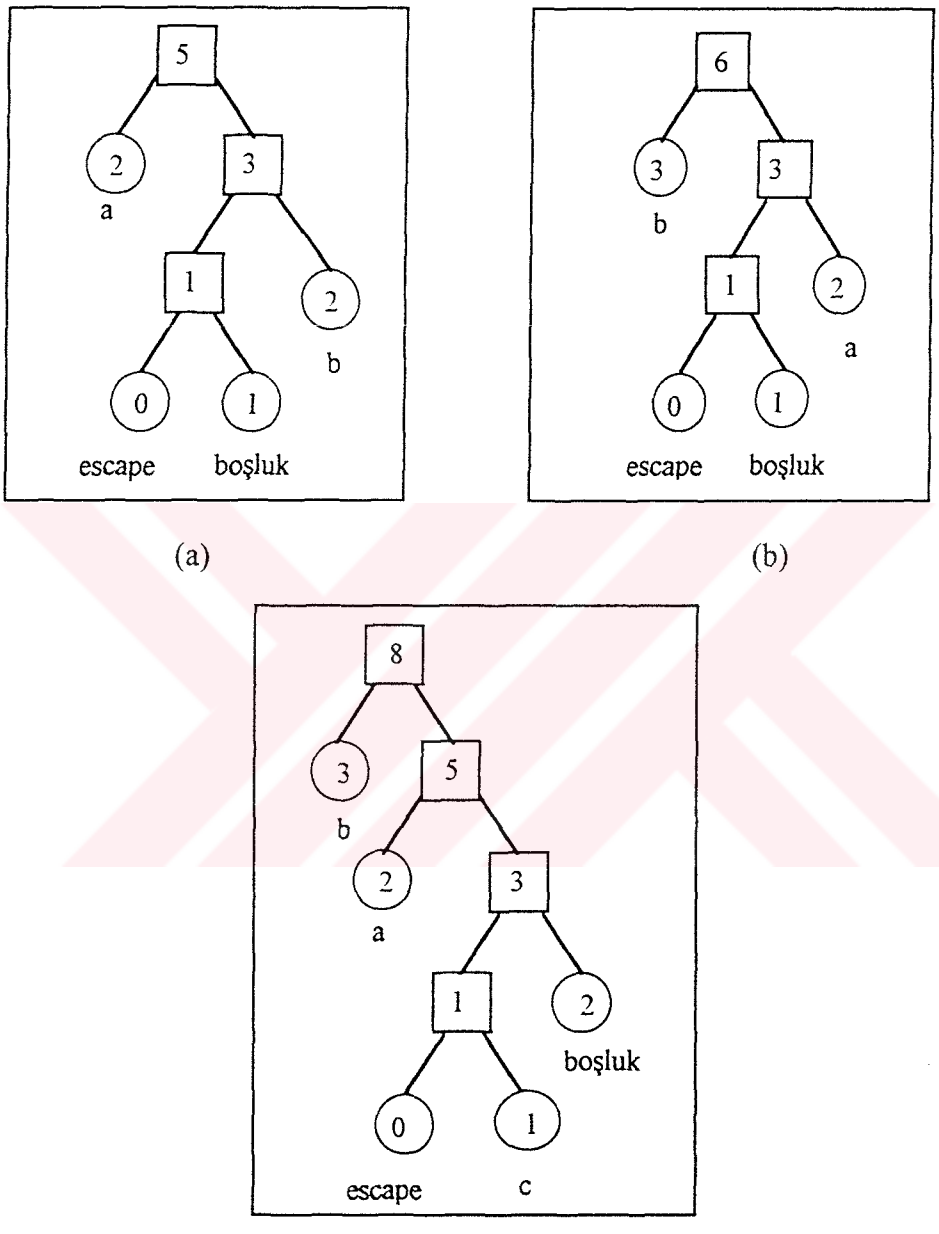
aynı sembolleri gösterebilmeleri için) ağacı aynı şekilde oluşturabilmeleri gerekir. Bu nedenle bu iki algoritmanın senkronize edildiği söylenebilir.

FGK algoritmasının temeli Galleger (Galleger,1978) tarafından geliştirilen **Kardeş Özelliği'**ne (*sibling property*) dayanır. Bir ağacın aynı seviyede olan, aynı düğüme bağlı, yan yana iki yaprağı kardeşlik özelliğine sahip demektir. Kullanım sıklıklarına göre soldan sağa, küçükten büyüğe, sıralı olan aynı seviyedeki yaprakların birisinde kullanım sıklığı artarsa bu yaprak sıralamadaki yerini almak üzere aynı seviyede yer değiştirebileceği gibi, bir üst seviyeye de çıkabilir. Dinamik Huffman kodlamasında, kodlamaya giren yeni bir sembol kullanım sıklığını değiştirirse ve değişiklik yaprağın bağlı olduğu kökü değiştirirse Huffman ağacı bu değişikliğe uyum göstererek değişir ki bu bir dinamik yapı demektir.

Herhangi bir anda, n elemanlı kaynak mesajının k ($k<n$) tanesi mesaj içinde kullanılmış olsun. Başlangıçta, kod ağacı bir tek 0-düğüm (*0-node*) olarak isimlendirilen tek bir yaprak düğümünden oluşur. 0-düğümü, kullanılmayan (n-k) mesajı göstermek için kullanılan özel bir düğümdür. İletilen her mesaj için her iki tarafta ilgili ağırlığı bir arttırır ve kardeş özelliğini muhafaza etmek üzere kod ağacını tekrar oluşturur. Oluşturulan Huffman ağacı (k+1) yapraktan oluşmaktadır. Bunlardan k tanesi gönderilen mesajı ve bir tanesi de 0-düğümünü gösterir.

- Alınan (t+1) mesajı, daha önceden görünen (k) mesajdan biri ise bu mesajla ilgili mevcut kodu yollar ve ilgili ağırlığı bir arttırarak ağacı tekrar oluşturur.
- Alınan (t+1) mesajı, daha önce görünmeyen bir mesaj ise 0-düğümünün kodu ve kullanılmayan (n-k) mesajdan hangisinin yollandığı bildirilir. Daha sonra 0-düğümü bir çift yaprak oluşturmak üzere bölünür. Bunlardan bir tanesi (t+1) mesajını diğeri 0-düğümünü oluşturur. Her düğümde ilgili mesajın görünme sayısı saklıdır. Düğümler kardeşlik özelliği sıralamasındaki pozisyonlarını ifade edecek şekilde numaralandırılırlar. Ağacın güncellenmesi (t+1) düğümünden köke doğru bir taraftan diğer tarafa yer değişikliği ile yapılır. Yer değiştirme işlemi (t+1) düğümü için ve köke doğru bütün bağlantılar için ağırlıkları bir arttırır. Düğümler, kardeş özelliğini sürdürmek üzere yer değiştirirler.

Örnek: "aa bb" mesajı yollandıktan sonraki ağaç Şekil 3.9(a)'daki halini alır. Üçüncü bir "b" gönderildiğinde, ağaç Şekil 3.9(b)'deki gibi olur. Arkasından bir boşluk gönderildiğinde ağacın durumunda bir değişme olmaz. Birinci "c" değeri gönderildiğinde ağaç Şekil 3.9(c) deki halini alır.



Şekil 3.9 Dinamik Huffman ağacı

3.3.1.2 Dinamik Aritmetik kodlama

Dinamik Aritmetik kodlamanın iki temel adımı vardır.

1. Aritmetik kodlamada sembollerin sırasının önemi yoktur. Kodlama sırasında tablodaki semboller yer değiştirebilir.
2. Bir X sembolünü kodlamak için kodlayıcıya sembolün kümülatif frekansları verilmelidir. Bu X'in frekansının, X her kodlandığında değiştirilebileceğini gösterir. Kodlayıcı ve kod çözücünün bunu nasıl yapacakları konusunda uzlaşmaları gerekir.

Bu iki özellikten yararlanarak dinamik aritmetik kodlamayı açıklayabiliriz. Kodlama algoritması iki bölümden oluşur; Model ve aritmetik kodlama.

Model bölümü, giriş dizisinden yeni bir sembolü okur ve kodlayıcıya bu sembol ve kodlama için gerekli iki kümülatif frekansı gönderir. Model bölümü daha sonra sembol sayısını bir arttırır ve kümülatif frekansı tazeler. Burada önemli olan nokta, model bölümünün sembolün olasılığını o sembolün eski sayısından oluşturması ve sembol kodlandıktan sonra sayıyı bir arttırmasıdır. Böylece kod çözücü bölümün kodlayıcıyı takip etmesi mümkün olabilir.

Model bölümü: sembolleri, onların kullanım sıklıklarını ve kümülatif frekanslarını bir dizide saklar. Bu dizideki sayıların sıralı olarak saklanması gerekir. Her seferinde bir sembol okunur ve onunla ilgili sayı bir arttırılır ve kümülatif frekans yenilenir. Daha sonra sıralı diziyi muhafaza etmek için gerekli yer değiştirme işlemi yapılır.

Bu durumda basit bir veri yapısı olan arama (*search*) ve güncelleme (*update*) işlemleri kullanılır. Bu yapı bir diziye yerleştirilmiş dengeli bir ağacı oluşturur (Dengeli ağaç tamamlanmış bir ikili ağaçtır, ancak bazı alt sağ düğümleri olmayabilir). Ağaçta alfabenin her karakteri için bir düğüm bulunmalıdır ve dengeli olduğu için ağacın yüksekliği $\lceil \mathrm{Log}_2 n \rceil$ olmalıdır. Burada n alfabenin boyutunu gösterir.

Çizelge 3.8 On sembollü bir alfabe ve sembollerin kullanım sayıları

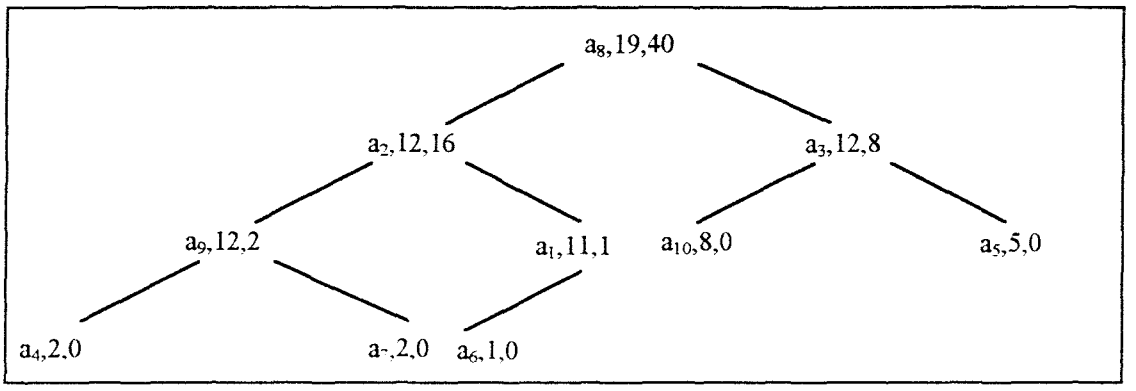
sembol	a_8	a_2	a_3	a_9	a_1	a_{10}	a_5	a_4	a_7	a_6
sıklık	19	12	12	12	11	8	5	2	2	1
küm.toplam	40	16	8	2	1	0	0	0	0	0

Çizelge 3.8'de sıklık ve kümülatif toplamları ile verilen sıralı bir dizi Şekil 3.10(a)'daki gibi dengeli bir ikili ağaca yerleştirilebilir. Ağacı oluşturmanın kolay bir yolu vardır. Dengeli bir ikili ağaç hiçbir işaretçiye (*pointer*) gerek kalmadan bir diziye yerleştirilebilinir. Dizinin ilk elemanı ağacın kökünü oluşturur. Dizinin i. elemanının iki yaprağı vardır ve bunlar soldan sağa dizinin (2i) ve (2i+1) indisli elemanlarıdır. Dizinin j. elemanının kökü $\lfloor j/2 \rfloor$ indisli elemandır.

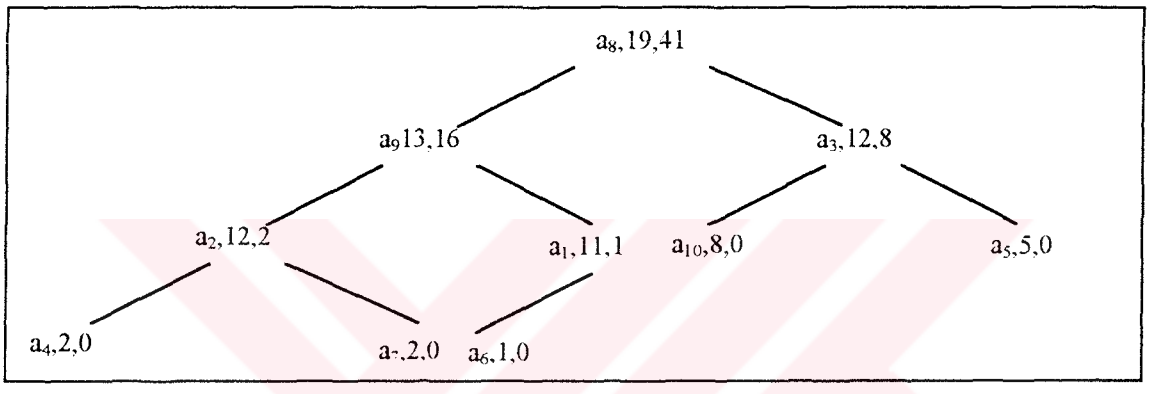
Ağaçdaki her bir düğüme bir sembol ve onun kullanım sıklığına ek olarak birde sol alt ağaçlardaki toplamı eklenir. Bu değer kümülatif frekansı hesaplamak için kullanılacaktır.

Giriş dizisinden okunan bir sonraki sembolün a_9 olduğunu kabul edelim. Bu sembolün sayısı 12'den 13'e bir arttırılır. Model bölümü diziyi sıralı olarak saklamak için, değeri a_9'dan daha küçük olan sayıyı arayarak gerekli yer değişikliğini yapar. Bu arama işlemini dizi fazla uzun değilse doğrusal (*linear*), uzunsa ikili arama (*binary search*) algoritması ile yapar. Yukarıdaki örnek için a_2 ve a_9 dizi üzerinde yer değiştirmelidir. Şekil 3.10(b)'de ağacın güncellenmiş şeklini gösterir.

Bir X sembolü için kümülatif frekansa gerek duyulduğunda Model bölümü kökten X sembolünün bulunduğu düğüme doğru dallanır ve yol üstündeki sayıları toplar. Verilen örnekte a_6 sembolünün kümülatif frekansı kökten a_6 sembolünün bulunduğu düğüme doğru 28'dir. a_6 sembolünün kümülatif frekansı için önce a_2'ye dallanılır ve 12 + 16 = 28 bulunur. Daha sonra a_6 düğümünün bulunduğu sol dal alınır fakat kümülatif frekansa ekleme yapılmaz. a_6 düğümüne ulaşıldığında sol alt ağaç sayısı 0, kümülatif frekansa eklenir. Böylece kümülatif frekans 28 olarak hesaplanmış olur Şekil 3.10(c).



(a)



(b)

a_4	2	0-1
a_9	12	2-13
a_7	2	14-15
a_2	12	16-27
a_6	1	28-28
a_1	11	29-39
a_8	19	40-58
a_{10}	8	59-66
a_3	12	67-78
a_5	5	79-83

(c)

Şekil 3.10 Dinamik Aritmetik kodlama

3.4 Sözlük Kullanan Yöntemler

Sözlük tabanlı sıkıştırma metodunda (Dictionary Method), istatistiksel yöntemler kullanılmaz. İstatistiksel sıkıştırma yöntemlerinde, verinin istatistiksel modeli kullanılır. Sözlük tabanlı sıkıştırma yöntemlerinde istatistiksel model ve değişken uzunluklu kodlar kullanılmaz. Bunun yerine sembollerden oluşan katarlar (*string*) kodlanıp kullanılır. Sözlükler, statik ve dinamik olarak ikiye ayrılır. Statik sözlüklere ekleme yapılabilir fakat silme yapılamaz. Dinamik sözlükte ise hem ekleme hem de silme yapılabilir. Statik sözlüğe verilebilecek örnek, herhangi bir dilde yazılmış metni sıkıştırmak için kulanılan o dile ait sözlüktür. Giriş biriminden gelen bir sözcük sözlük içerisinde aranır, bulunursa çıkış birimine o kelimenin sözlükteki sırası yerleştirilir aksi halde sözcük sıkıştırılmadan, aynen yazılır. Çıkış biriminde hem her kelimeye karşı gelen kodlar hem de kodlanmış bir kelimenin karakterleri bulunabileceğinden bunları birbirinden ayırmak gerekir. Bunun için yazılan her bir sözcük için fazladan bir bit (*işaret biti*) kullanılmalıdır. Bu bit, 0 değerini almışsa sözcüğün sözlükte bulunduğunu, 1 değerini almışsa ardından gelecek olan sözcüğün byte uzunluğu ve kendisinin kodlanacağını gösterir. 512 bin kelimeden oluşan bir sözlük düşünelim, bu sözlüğü kodlamak için işaret bitiyle birlikte 20 bit'e ihtiyaç vardır. Örneğin, sözlükte "benek" kelimesini arayıp 1031. sırada bulmuşsak bu kod 0|00000000010000000111 şeklinde iken "set" sözcüğünün sözlükte bulunmadığını kabul ettiğimizde kodlama 1|0000011|01110011|01100101|01110100 şeklinde gerçekleşir. Ortalama sözcük uzunluğunu 5 karakter kabul edersek, kodlama için 48 bit'e ihtiyacımız olacaktır. 48 bit'i, 20 bit'le kodlayabiliyorsak iyi bir sonuç elde etmişiz demektir. Bu sonuca ulaşabilmek için, metin içerisinde geçen kaç kelimenin sözlükte bulunması gerekir? Bu işlemin olasılığına P, sözlükte bulunan sözcük sayısına N dersek çıkış biriminin boyu, $N[20P + 48(1 - P)] = N[48 - 28P]$ bit olacaktır. Her sözcüğün ortalama 5 karakterden oluştuğunu kabul edersek giriş biriminin uzunluğu 40N bit olacaktır.

3.4.1 Katar (*string*) sıkıştırma

Sıkıştırma yöntemlerinde, tek bir sembolün sıkıştırılmasındansa sembollerden oluşan bir katarın sıkıştırılması daha etkilidir. İki elemanlı bir alfabemiz $\sum = \{a_1, a_3\}$ ve bunların olasılıkları P_1=0.8, P_2=0.2 verilmiş olsun. Olasılıkların ortalaması da 0.5 olup, varyansı $|0.8 - 0.5| + |0.2 - 0.5| = 0.6$ olarak bulunur. İki elemanlı bir alfabenin her bir elemanı 1 bit ile kodlanacaktır. Çizelge 3.9' daki alfabede her elemanın iki sembolden oluştuğu, Çizelge 3.10'daki alfabenin de her elemanının üç sembolden oluştuğu zamanki değerleri verilmektedir.

Çizelge 3.9 Her elemanı iki sembolden oluşan bir alfabenin Huffman kodları ve olasılıkları (Salamon, 1998)

Katar	Olasılık	Kod
a_1a_1	0.8 x 0.8 = 0.64	0
a_1a_2	0.8 x 0.2 = 0.16	11
a_2a_1	0.2 x 0.8 = 0.16	100
a_2a_2	0.2 x 0.2 = 0.04	101

Çizelge 3.10 Her elemanı üç sembolden oluşan bir alfabenin Huffman kodları ve olasılıkları (Salamon, 1998)

Katar	Olasılık	Kod
$a_1a_1a_1$	0.8 x 0.8 x 0.8 = 0.512	0
$a_1a_1a_2$	0.8 x 0.8 x 0.2 = 0.128	100
$a_1a_2a_1$	0.8 x 0.2 x 0.8 = 0.128	101
$a_1a_2a_2$	0.8 x 0.2 x 0.2 = 0.032	11100
$a_2a_1a_1$	0.2 x 0.8 x 0.8 = 0.128	110
$a_2a_1a_2$	0.2 x0.8 x 0.2 = 0.032	11101
$a_2a_2a_1$	0.2 x 0.2 x 0.8 = 0.032	11110
$a_2a_2a_2$	0.2 x 0.2 x 0.2 = 0.008	11111

Çizelge 3.11'deki sonuçlar, sembollerden oluşan bir katarı kodlamanın, tek bir sembolü kodlamaktan daha verimli olduğunu, bu yüzden de sözlük tabanlı sıkıştırma yöntemlerinin diğer sıkıştırma yöntemlerine göre neden tercih edildiğini göstermektedir.

Çizelge 3.11 Sembol uzunluklarına göre varyans ve ortalama kod uzunlukları

Sembol uzunluğu	Olasılıkların varyansı	Ortalama kod uzunluğu
1	0.6	1
2	0.78	0.78
3	0.792	0.728

3.4.2 Lempel_Ziv algoritmaları

Lempel ve Ziv (Lempel ve Ziv, 1977), LZ77 ve LZ78 olarak bilinen iki algoritma geliştirmişlerdir. Daha sonraki dinamik algoritmaların gerçekleştirilmesinde öncülük edip LZXX türevlerinin ortaya çıkmasına neden olmuşlardır. Bu iki yöntem, bir sözlüğe sözcük veya ibareler (*phrase*) girilmesi, sıkıştırılacak metin içerisinde bu sözlükte bulunan sözcük veya ibarelerin tekrar edildiği görüldüğünde çıkış olarak o sözcüğün veya ibarenin yerine sözlükdeki sırasının verilmesi esasına dayanmaktadır.

3.4.2.1 LZ77 (*kayan pencere - sliding window*)

LZ77 algoritması, daha önceden okunmuş olan giriş bilgisini sözlük olarak kullanır. Kayan pencere tabanlı adı verilen bu yöntemde, pencere iki parçaya ayrılmıştır. Sol taraftaki pencere arama alanı (*search buffer*), sağ taraftaki pencere de kaynak kod *alanı (look-ahead buffer*) olarak adlandırılır. Arama alanı, mevcut olan sözlük ve son olarak okunup kodlanmış bilgileri, kaynak kod alanı da henüz kodlanmamış giriş bilgilerini içerir. Kaynak kod alanının boyu, arama alanının boyundan kısa olmak zorundadır. LZ algoritmasında arama alanı büyüdükçe sıkıştırma oranı artar buna karşılık sıra belirleyicisinin (*offset*) kodlanması için gerekli olan bit sayısı arttığı gibi arama için geçen zaman ve aranan sözcüğün bulunması için gerekli olan çevrim sayısı da artmaktadır. LZ77'de belirteç (*token*) üç parçadan oluşur, sıra belirleyicisi, uzunluk ve gelecek semboldür. Arama ve kaynak kod alanı Şekil 3.11'deki gibi verilmiş olsun.

tek□tez□kopyası	□tezimin□olmalı

Şekil 3.11 Arama ve kaynak kod alanı

Kaynak kod olanındaki "□", arama alanında kodlayıcı tarafından sağdan sola doğru taranır. İlk olarak 8.sırada bir eşleşme bulunur ve geriye doğru aramaya en uzun eşleşme bulununcaya kadar devam edilir. İkinci kez 12.sırada 4'lü bir eşleşme bulunur ve kodlayıcı bu bilgiyi çıkış birimine (12,4,"i") olarak gönderir. Bu yöntemde en son eşleşme değerlendirilir. Bunun dezavantajı sıra belirleyicisinin (offset) kodlanacak bit sayısını arttırır. En yakındaki eşleşme alınırsa, algoritma daha karışık olur fakat buna karşılık sıra belirleyicisinin kodlanacak bit sayısı azalır.

Örnek:

tek□tez□kopyası□	→ (0,0,"t")
tek□tez□kopyası	→ (0,0,"e")
tek□tez□kopyası	→ (0,0,"k")
tek□tez□kopyası	→ (0,0,"□")
tek□tez□kopyası	→ (4,2,"z")
tek□tez□kopyası	→ (4,1, "k")

Temel LZ77 yöntemi, 80'li ve 90'lı yıllarda birçok araştırmacı tarafından geliştirilmiştir. Bu geliştirmelerden ilki, belirteç'in ilk iki alanının (sıra belirleyicisi ve uzunluk) değişken uzunlukta olması, ikincisi kaynak kod ve arama alanlarının boyunun uzatılması (arama alanının boyunun uzatılması ile eşleşmelerdeki başarının yükselmesine rağmen arama için geçen süre artmıştır) ve üçüncüsü kayan pencere ile ilgilidir. Çakışan her elemandan sonra arama alanının kaydırılmasında dairesel kuyruk (*circular queue*) uygulanmıştır.

Arama alanının boyunu belirlemek ve eşleşen sözcüğün bulunması için gerekli algoritmaların belirlenmesi birçok çalışmaya kaynak olmuştur (Arama alanının uzunluğu S ise, sıra belirleyicisinin bit uzunluğu $\lceil \log_2 S \rceil$ olacaktır).

3.4.2.2 LZ78 algoritması

LZ78 algoritması, LZ77 algoritmasında olduğu gibi kayan pencere, arama ve kaynak kod alanı kullanmaz. Bunların yerine önceden karşılaştırılmış sözcüklerden oluşan bir sözlük kullanır. Boş olarak başlayan sözlüğün boyu kullanılabilir hafıza ile sınırlıdır. Kod çözücünün kullandığı belirteç'in, sözlük sıra belirleyicisi ve sözcüğün kodunu tutan iki alanı vardır. Sözlük boş (*null*) bir karakter ile başlar ve sıra belirleyicisinin değeri sıfırdır. Daha sonra giriş biriminden gelen *w* sembol sözlükte aranır, çakışma yoksa bir sonraki sıraya, sıra belirleyicisi "0" değerini alarak yerleşir, çakışma olmuşsa bir sonraki sembol *k*, bir önceki sembole eklenerek *wk* oluşturulur ve sözlükte arama işlemine devam edilir. Bu işlem en uzun sözcük sözlükte bulunana kadar tekrar eder. *wk* sözlükte bulunamazsa, sözlüğün en son sırasına yerleşir ve çıkışa giden belirteç'in ilk alanına en son çakışmanın olduğu sıra, ikinci alanına da *k* yerleştirilir. Çizelge 3.12 aşağıdaki örnek için gösterilmiştir.

Örnek:	tek□tez□kopyası□tezimin□olmalı

Çizelge 3.12 LZ78 algoritmasının çalışması

sıra	sözlük	belirteç	sıra	sözlük	belirteç
0	null				
1	"t"	(0,"t")	8	"o"	(0,"o")
2	"e"	(0,"e")	9	"p"	(0,"p")
3	"k"	(0,"k")	10	"y"	(0,"y")
4	"□"	(0,"□")	11	"a"	(0,"a")
5	"te"	(1,"e")	12	"s"	(0,"s")
6	"z"	(0,"z")	13	"ı"	(0,"ı")
7	"□k"	(4,"k")	14	"□t"	(4,"t")

LZ78 algoritmasında, sözlüğe ekleme işlemi sözlük dolana kadar devam ettirilir ve sonraki sözcükler sözlüğe eklenmeden kodlanır (*FROZEN yöntemi*). LZ78'in türevlerinde yeni öneriler getirilmiştir. Bunlardan biri, iki sözlük kullanmaktır. Sözlüklerden biri dolunca diğeri ile işleme devam edilir. Bir diğeri ise, sözlük dolduğunda en az kullanılan sözcüğü çıkarıp yenisini eklemektir.

3.4.2.3 LZW algoritması

LZW yöntemi, Ziv ve Lempel tarafından geliştirilmiş ve T.Welch (Welch, 1984) tarafından değiştirilmiştir. LZW'de belirteç'in ikinci alanı çıkarılmıştır. LZW'nin belirteçi, sözlük sıra belirleyicisinden oluşmaktadır. Bu yöntemi daha iyi anlayabilmek için geçici bir süre için ağaç yapısındaki sözlüğü unutup, değişken uzunluktaki bir katar dizisi olarak düşünelim. LZW yöntemi, sözlüğe giriş biriminden gelen bir sözcük eklemeden önce alfabenin bütün elemanlarından oluşan 256 adet giriş bilgisi* ile başlar ve kodlayıcıya gelen giriş bilgisini *I* 'ya yerleştirip sözlük içerisinde arar. Bulduğunda çıkışa sözlük sıra belirleyicisi ile birlikte gönderir ve gelecek olan *x* sembolünü, *I* ile birleştirip, *Ix*'i sözlükte aramaya devam eder. Sözlükte bulamazsa sözlüğün sonuna *Ix* i ekleyip çıkışa *Ix* ve sözlük sıra belirleyicisini gönderir. Çizelge 3.13 aşağıdaki örnek için oluşturulmuş olan sözlük olup bu örnek için oluşan çıkış 116(t), 101(e), 258(k), 32(□), 256(te), 120(z), 32(□),258(k),112(o), 113(p), 121(y), 97(a), 115(s), 104(ı), 259(□t), 101(e), 120(z), 105(I), 109(m), 105(i), 110(n), 32(□), 112(o), 259(l), 109(m), 97(a), 259(l), 104(ı) olur.

Örnek:	tek□tez□kopyası□tezimin□olmalı

Çizelge 3.13 LZW sözlüğü

0 NULL	109 m	259 □t	271 ez
.....	110 n	260 tez	272 zi
32 SP	.....	261 z□	273 im
.....	115 s	262 □k	274 mi
97 a	116 t	263 ko	275 in
98 b	.....	264 op	276 n□
99 c	121 y	265 py	277 □o
100 d	.....	266 ya	278 ol
101 e	255 255	267 as	279 lm
.....	256 te	268 sı	280 ma
107 k	257 ek	269 ı□	281 al
108 l	258 k□	270 □te	282 lı

* ASCII tablosundaki 0-255 arası karakterler

3.4.3 Zip ve Gzip Algoritmaları

Zip ve Gzip algoritmaları, "sıkıştırma" algoritması olarak bilinen LZ77'nin bir türevi ile statik Huffman metodunun birleşiminden oluşur. 32 Kbyte'lık bir kayan sözlük ve 258 byte'lık kaynak kod alanı kullanır. Eğer bir katar sözlükte bulunmazsa olduğu gibi kodlanır.

Giriş verisi, kodlayıcı tarafından bloklara ayrılır. Blok uzunlukları keyfidir, yalnızca sıkıştırılamayan bloklar 64 Kbyte ile sınırlandırılmıştır. "Sıkıştırma" kodlayıcısı, Huffman ağacıyla yeni bir bloğa başlamanın uygun olacağına karar verdiği zaman, bir önceki blok sona erer. Harfler ve eşleşme uzunlukları Huffman ağacına, eşleşme uzaklıkları da diğer bir ağaca sıkıştırılır. Ağaçlar, herbir bloğun başında sıkıştırılmış olarak saklanır. Huffman ağaçları, herbir blok için önceki ya da sonrakilerden bağımsızdır.

Tekrarlanan katarlar çırpı (*hash*) fonksiyonları yardımıyla bulunur. 3 karakter uzunluklu giriş katarları, çırpı tablolarına girilir. Bir sonraki 3 byte için çırpı göstergeç'i (*index*) hesaplanır. Bu gösterge için çırpı zinciri boş değilse, zincirdeki tüm katarlar o anki giriş katarıyla karşılaştırılıp en uzun eşleşme seçilir.

Çırpı zinciri, en yeni oluşturulmuş katardan başlayarak taranır. Böylece küçük uzaklıklara göz yumularak Huffman kodlamasının avantajlarından yararlanılır. Çırpı zincirleri tek başına eklenir. Çırpı zincirinden iptal edilme yoktur, algoritma çok eski eşleşmeleri siler.

En kötü durum olasılığını ortadan kaldırmak için, çok uzun çırpı zincirleri çalışma zamanı opsiyonlarında belirlenen bir değere göre kesilerek kısaltılır. Sonuç olarak, "sıkıştırma" her zaman en uzun eşleşmeyi bulmayabilir ama yeteri kadar uzun bir eşlik yakalar.

"Sıkıştırma", eşliklerin seçimini bir değerlendirme mekanizması yoluyla yapar. N uzunlukta bir eşleşme yakalandığı zaman, "sıkıştırma" bir sonraki giriş byte'ında daha uzun bir eşleşme arar. Eğer daha uzun bir eşleşme yakalanırsa, bir önceki eşleşmenin uzunluğu bire düşürülür ve daha uzun olan eşleşme çıkışa verilir. Aksi halde, orijinal eşleşme saklanır ve bir sonraki eşleşme araması N adım sonra gerçekleşir.

Eşleşme değerlendirme mekanizması, çalışma zamanı parametreleri ile de kontrol edilebilir. Eğer mevcut eşleşme yeterince uzunsa, "sıkıştırma" daha uzun bir eşleşme için yapılan aramayı düşürür ve işlemin tamamını hızlandırır. Eğer sıkıştırma oranı hızdan daha önemliyse, "sıkıştırma" ilk eşleşme yeterince uzun olsa bile ikinci bir arama yapar.

Çok hızlı sıkıştırma modlarında, eşleşme değerlendirme mekanizması kullanılmaz. Bu hızlı modlarda, ancak hiçbir eşleşme bulunmaması ya da eşleşmenin çok uzun olması durumlarında yeni katarlar çırpı tablolarına yerleştirilir. Bu durum sıkıştırma oranını düşürür ancak daha az yerleştirme ve daha az taramayla zaman tasarrufu sağlanır.

3.4.4 ARC ve PKZIP algoritmaları

ARC, Robert A.Freed tarafından 1980'lerin ortalarında geliştirilmiş bir sıkıştırma/ arşivleme/ kataloglama programıdır (Salomon,1998). Gerek iyi bir sıkıştırma tekniği olması, gerekse birden çok dosyayı tek bir arşiv dosyasında toplayabilmesi açısından, PC kullanıcıları arasında kısa zamanda yaygınlaşmıştır. Arşivlemenin çok kullanışlı olduğu iki durumu gözden geçirelim:

1. Bir grup dosya modemler arasında iki yönlü olarak transfer edilecektir. Herbir dosyanın transferi sırasında iki modem de bir protokol yardımıyla veri iletimi yapacakları için bu dosyaları arşivleyip tek bir büyük dosya olarak göndermek, iletim zamanında ciddi bir kazanç sağlayacaktır.
2. Disk ne kadar büyükse minimum blok uzunluğu da o oranda artacağından, yüzlerce küçük dosyayı büyük bir sabit-disk üzerine koymak gereksiz yer tüketimine yol açacaktır. Örneğin, 500 Mbyte disk için minimum dosya uzunluğu 5 Kbyte olabilir. Bu durumda, herbiri 500 byte olan 100 küçük dosya düşünelim. Bu 100 dosya, minimum dosya uzunluğu baz alındığında disk üzerinde 500 Kbyte yer kaplayacaktır. Oysa, bir arşive yerleştirildiğinde aynı bilgi 50 Kbyte yer kaplayacaktır.

Pekçok modern arşivleyici, kendi kendini çözümleme yeteneğine sahiptir. Bu tarz arşivleyiciler, sıkıştırılmış dosyanın yanında bir de sıkıştırma çözümleyicisi içerirler. Böylelikle dosya bir miktar daha büyür ama kendi kendini çözümleyebilir.

3.4.5 EXE sıkıştırıcıları

LZEXE algoritması, F.Bellard(1980) tarafından EXE dosyalarını (*PC de yürütülebilir dosyalar*) sıkıştırmak üzere yazılmış özel amaçlı programlardır. Buradaki düşünce LZEXE tarafından sıkıştırılmış bir EXE dosyasının tek bir komutla açılması ve aynı zamanda yürütülmesidir. Bu teknikte kod çözücü, açılan dosyayı disk üzerine yazmaz, belleğe yükleyip adresleri yeniden düzenleyerek programı çalıştırır. Kod çözücü program, açılan program tarafından kullanılan belleği kullanacağı zaman ayrıca bir bellek gereksinimi duymaz. Kod çözücü program büyüklüğünün, kendiliğinden açılan arşiv kod çözücülerine göre küçük olduğu gözlenir.

Programın algoritması LZ tekniğine benzemektedir. Bu algoritma, katar eşleşmelerini algılamak için dairesel kuyruk yapısını ve bir sözlük ağacını kullanır. Eşleşmenin yeri ve boyutu Huffman yöntemine dayalı yardımcı bir algoritma ile kodlanır. Sıkıştırılmayan byte'lar değiştirilmeden muhafaza edilir. Bunun nedeni bu verileri daha fazla sıkıştırmaya çalışmanın daha karmaşık ve büyük bir kod çözücü gerektirmesidir. Kod çözücü, sıkıştırılmış EXE dosyasının sonuna yerleştirilir ve 330 byte uzunluğundadır. LZEXE tekniği ile sunulan EXE sıkıştırması, kullanıcılar ve yazılım geliştiricileri için çok kullanışlı sıkıştırıcı programlarıdır.

3.4.6 Burrows-Wheeler metodu

M.Burrows ve D.Wheeler (Burrows ve Wheeler,1983) tarafından bulunmuş bir yöntemdir. Bu yöntem de sıkıştırma metodlarının çoğu gibi veri akış yolu (*streaming*) modunda çalışır. Bu modda, kodlayıcı-çözücü (*codec*) bir ya da birden çok byte işler. Bu işlem dosya sonu

(*end-of-file*) oluncaya kadar devam eder. Burrows-Wheeler (BW) metodu, blok modda çalışır. Giriş verisi bloklar biçiminde okunur ve herbir blok ayrı katarlar şeklinde kodlanır. Bu sebeple BW metodu, "blok listeleme" adıyla da bilinir. BW metodu genel amaçlıdır. Görüntü, ses ve yazı üzerinde çok mükemmel sonuçlar verir ve çok yüksek sıkıştırma oranlarına ulaşır (1:8, ve üzeri).

BW metodunun ana mantığı, n sembollü bir S katarını aşağıdaki iki koşulu sağlayan bir L katarına aktarmak şeklindedir:

1. L katarının herhangi bir alanı, birkaç sembolün birleşiminden oluşmuştur. Bir başka deyişle, L' nin herhangi bir pozisyonunda bir s sembolü varsa, diğer s 'ler de büyük ihtimalle onun yakınında toplanmış olacaktır. Bu demektir ki, Move-to-Front metodu gerekirse RLE ile birleştirilip kullanılırsa, L kolaylıkla sıkıştırılabilinir. Buradan çıkan sonuç: BW metodu, eğer n değeri büyükse iyi sonuç verecektir (Her bir katarda en azından birkaç bin sembol olması durumunda).
2. L'yi kullanarak, orijinal S katarını yeniden oluşturmak mümkündür (Yeniden oluşturmak için L 'nin yanı sıra bir T dizisine ve I degerine ihtiyaç vardır).

Sembollerin aktarımının matematiksel ifadesi permutasyondur ve n sayıda sembol içeren bir katarın n! kadar permutasyonu olacaktır. Bu rakam, n'nin küçük değerleri için bile oldukça büyük sayılır. Bu yüzden BW'nin kullanacağı permutasyon dikkatle seçilmelidir. BW kodlayıcı-kodçözücü (*codec*) şu adımları takip eder:

1. L katarı, şifreleyici tarafından S'nin bir permutasyonu olarak yaratılır. Kod çözücü tarafından kullanılmak üzere, I ve T ile ifade edilen diğer bazı bilgiler de yaratılır.
2. Kodlayıcı L,T ve I'yı sıkıştırarak sonuçları çıkış verisine atar. Bu adım genellikle RLE ile başlar, Move-to-Front kodlamasıyla devam eder ve Huffman kodlamasıyla da sona erer.
3. Kod çözücü, çıkış verisini okur ve 2. adımdaki yöntemlerin aynısını tersten uygulayarak kodu çözer. Sonuç; L,T katarı ve I değişkenindedir.
4. L,T ve I kod çözücü tarafından okunur ve orijinal katar yeniden oluşturulur.

İlk adım, *L* katarının *S*'den nasıl yaratıldığını ve geri dönüşüm için *T* ve *I*'ya hangi bilgilerin yüklendiğini anlamaktır.

Bir örnek üzerinde inceleyelim. Beş elemanlı ve elemanları A={s,ı,n,a,v} olan bir A kümesi verilsin. Kodlayacağımız *S* katarında da "sınav" kelimesi bulunsun. Kodlayıcı, ilk satırına *S* katarını yerleştirdiği (n*n) boyutunda bir matris üretir ve aşağıya doğru diğer satırlara *S*'nin soldan itibaren bir karakterinin kaldırılıp, katarın sonuna eklendiği bir dönme (rotate) mekanizmasıyla (bir sembol sola shift) *S*'nin (n-1) tane kopyasını yerleştirir (Şekil 3.12).

s_0	s ı n a v
s_1	ı n a v s
s_2	n a v s ı
s_3	a v s ı n
s_4	v s ı n a

Şekil 3.12 Permütasyon matrisi

Daha sonra matris, satıra göre sözlüksel karşılaştırma (*lexicographic*) yöntemiyle sıralanan yeni bir matris üretilir. Her iki matrisin satırları ve sütunları *S*'nin permütasyonlarıdır ve *S*'nin tüm sembollerini içerir. Kod çözücü tarafından seçilen *L*, sıralı matrisinin en son kolonudur (Şekil 3.13).

s_3	a v s ı n
s_1	ı n a v s
s_2	n a v s ı
s_0	s ı n a v
s_4	v s ı n a

Şekil 3.13 Sıralı matris

S'yi, *L*'den tekrar oluşturabilmek için kullanılacak diğer bilgiler, kodlayıcı tarafından oluşturulan *T* dizisi ve sıralı matriste orijinal katarın bulunduğu satırın numarasıdır ki buda *I* değişkeninde tutulur. *T* dizisini oluştururken izlenen yol; orijinal katarın matristeki satır

numarası (verilen bu örnek için 3'tür), *T* dizisinin 3. indisinin, bir sonraki katarın ,(s_1)'in, matristeki satır değerini alması ile başlar (verilen bu örnek için "1" dir). *T* dizisinin 1. satırına da s_1'den sonra gelen katarın, s_2, matristeki satır numarası yazılır. İşlem devam ettirildiğinde Şekil 3.14 elde edilmiş olur. *L*,*T* dizileri ve *I* değişkeninin değerleri sırasıyla RLE, Move-to-Front ve Huffman kodlaması ile arka arkaya sıkıştırılır. Kodlanmış veriyi çözerken yukarıdaki yöntemlerin aynısı tersten yapılır.

sıra			T	L	
0	s_3	a	4	n	s_3
1	s_1	ı	2	s	s_1
2	s_2	n	0	ı	s_2
3	s_0	s	1	v	s_0
4	s_4	v	3	a	s_4

Şekil 3.14 T dizisinin oluşturulması

L ve *T*'den *S*'ye geri dönerken, (n-1) kez, *S* katarının elemanlarını bulmak için eşitlik (3.4) ve yeni I değerini hesaplamak için de eşitlik (3.5)'den yararlanılır (Şekil 3.15).

L[T[I]] (3.4)

I=T[I] (3.5)

I	L[T[I]]
1	s
2	ı
0	n
4	a
3	v

Şekil 3.15 L dizisinin geri elde edilmesi

BW yöntemi içerisinde bahsedilen Move-toFront yöntemini bir örnek ile açıklayalım. Elimizdeki *L* dizisinin her elemanı sırasıyla "A" alfabesi içerisindeki yerini çıkışa gönderip, alfabe içerisindeki yerinden alfabenin başına taşınır. Bu işlem n kez tekrar edilir. Geri çözme işleminde de giriş ve çıkış verileri yer değiştirerek yapılan işlemin tersi uygulanır (Şekil 3.16).

L	alfabe	code
n	sınav	2
s	nsıav	1
ı	snıav	2
v	ısnav	4
a	vısna	4

code	alfabe	L
2	sınav	n
1	nsıav	s
2	snıav	ı
4	ısnav	v
4	vısna	a

Şekil 3.16 Move-to-front yöntemi ile kodlama ve kod çözme

3.4.7 Kelime tabanlı yazı sıkıştırma

Buraya kadar incelenen tüm sıkıştırma yöntemleri, küçük alfabeler üzerinde çalışır. Tipik bir alfabe, iki binary dijit, 16 tane 4-bitlik pixel, 7-bit ASCII kod veya 8-bit byte içerir. Kelime içeren büyük alfabeler için kullanılan bir yöntemdir. Bu yöntem giriş verisinin yazı (*text*) olması durumunu kapsamaktadır. Bir yazı verisi akışında kelime, ya sayısal olmayan karakterler katarı (*harf ya da sayı*) ya da noktalama işaretleri, boşluk vb. diğer karakterleri belirtir. Sayısal olmayan tüm sözcüklere A, diğerlerine de P diyelim. Herhangi bir yazıda (bir program kodu) A ve P değişen biçimlerde içiçe ve ayrı kullanılabilir.

Örnek olarak:

' for (short j=0; j<npoints; j++)' kod satırı 15 sözcükten oluşmaktadır.

' ', 'for', ' (', 'short', ' ', 'j', '=', '0' , '; ', 'j', ' < ', 'npoints', '; ', 'j', '++).'

3.4.8 Kelime tabanlı Huffman kodlaması

Bu yöntem, karakter tabanlı uyarlanabilir Huffman kodlamasının biraz değiştirilmiş şeklidir. A ve P alfabeleri için iki Huffman ağacı oluşturulur ve algoritma bunlar arasında değiştirilir. Bu yöntemde karşılaşılacak sorunlar şöyle sıralanabilinir.

1. Yeni kelimelerin, hangi formatta çıkış yapacaklarına karar vermek gerekir. Yeni kelimeler çıkış verisine kendi karakterlerinin ASCII kodlarını kullanarak escape kodun hemen yanında yazılırlar. Ancak, normal olarak bir kelime birçok karakter içereceğinden, kelimeyi karakter tabanlı uyarlanabilir Huffman metodunu kullanarak kodlamak daha uygun olacaktır. Kısaca, kelime tabanlı uyarlanabilir Huffman algoritması, zaman zaman kullanılan karakter tabanlı uyarlanabilir Huffman algoritması içermektedir. Çünkü, kısa bir giriş verisi oldukça yüksek bir yüzdede yeni kelime içerebilir. Çıkış verisinde bunların ham kodlarını yazmak, sıkıştırma performansını önemli bir oranda düşürür.

2. Geniş Huffman ağaçları sebebiyle, bellek sorunu yaşanabilir. Bunun için en iyi çözüm, tüm ağacı silmek yerine uygun dalları iptal edip yeni bir Huffman ağacı düzenlemektir.

   Yapılan çalışmalar göstermiştir ki, kelime tabanlı uyarlanabilir Huffman kodlaması, karakter tabanlı uyarlanabilir Huffman kodlamasına göre daha iyi bir sıkıştırma başarımı alınmasına neden olmaktadır. Ancak, Huffman ağacı genişleme eğiliminde olduğu için daha yavaştır.

3.4.9 Kelime tabanlı LZW algoritması

Bu yöntem, karakter tabanlı LZW metodunun benzeridir. Giriş verisindeki kelime sayısı önceden bilinmediğinden, tahminlerin çok üstünde olabilir. LZW sözlüğü karakter tabanlı orijinal LZW metodunda olduğu gibi tüm olası kelimeler için baştan tanımlanmaz. Boş bir sözlükle başlanır (A ve P için ayrı sözlükler) ve escape kodu kullanılır. Sözlüğe eklenen her ibare, A ve P'den alınmış iki ayrı katar içerir. İlk katarı A olan tüm ibareler A sözlüğüne, ilk

katarı P olan tüm ibareler de P sözlüğüne yerleştirilir. İki ayrı sözlük olmasının avantajı, her iki sözlükte de ibarelerin 1'den başlayarak numaralanması ve böylece daha kısa bir numaralandırma sağlanmasıdır. Bu durumda iki sözlükte aynı numaraya karşı gelen farklı ibareler yer alacaktır. Ancak kod çözücü, çözülecek kodun hangi sözlükten geldiğini bildiği için bu durum sorun yaratmayacaktır.

4. DOSYA SIKIŞTIRMA ve TÜRK DİLİNİN YAPISI İLE İLGİLİ YAPILAN ÇALIŞMALAR

4.1 Dosya Sıkıştırma Üzerine Yapılan Çalışmalar

Günümüzde veri sıkıştırma üzerine yapılan çalışmalar yoğun bir şekilde devam etmektedir. Yeni algoritma önerilerinin yanı sıra bilinen algoritmaları *iyileştirme (sıkıştırma başarımını ve işlem hızını arttırma)* üzerine de çalışmalar yapılmaktadır. Huffman, Lempel ve Ziv'in sundukları algoritmalar üzerinde çalışmalar yapılıp, bu algoritmaların türevlerinin sunulduğu algoritmaların yanında yeni çalışmalarda sürdürülmektedir. Yazılım ile gerçekleşen sıkıştırma algoritmalarının donanım ile de gerçekleşmesi üzerine çalışmalar yapılmış ve yapılmaktadır.

Shannon-Fano (1949) ve Huffman (1952), sıklık bilgilerinden yararlanarak bir kodlama ağacı oluşturup, kullanım sıklığı büyük olan sembollere kısa kodlar veren bir kodlama tekniği geliştirmişlerdir. Her iki algoritmada ağacın oluşturulması sırasında farklı yöntemler kullanmışlardır.

Sıklığa bağlı tekrarları azaltan bir diğer algoritma da Aritmetic Coding yöntemidir. Elias tarafından önerilmiş, Abramson (1963) tarafından da geliştirilmiştir. Bu yöntemde kodlanacak veri 0 ile 1 aralığında ifade edilmektedir.

Ziv ve Lempel tarafından LZ77 (1977), LZ78 (1978) metin içerisindeki tekrarları azaltan, dinamik (*zaman içerisinde kodu güncelleyen*) bir algoritma geliştirilmiştir.

Robert A.Fred (1980) tarafından gerçekleştirilen **ARC**; P.Katz tarafından gerçekleştirilen **PKXX**; F.Bellard (1980) tarafından gerçekleştirilen EXE sıkıştırıcıları (*exe programlarını sıkıştırmak için yazılmış*) yaygın programlardır.

Ziv ve Lempel, (LZxx) algoritmalarını sunmalarından sonra bu yöntemler temel alınarak ya yeni algoritmalar önerilmiş yada bu algoritmaların hız ve sıkıştırma başarımını arttırmak için türev algoritmaları geliştirilmiştir. Bunlardan bazıları :

Storer ve Szymanski (1982) tarafından geliştirilen **LZSS**'dir. LZ77 yöntemini üç farklı yönüyle geliştirmişlerdir. Birincisi, kaynak kod alanını (*look-ahead buffer*) dairesel bir çevrim (*circular queue*), ikincisi arama alanını (*search buffer*) ikili arama ağacı (*binary search tree*) ve üçüncüsü kodlanan verinin üç ayrı alan ile değilde iki ayrı alan ile tutulmasını önermişlerdir.

A.S.Fraenkel, M.Mor ve Y.Perl (Fraenkel vd., 1983) tarafından yayınlanan makalede, İngilizce ve İbranice dilindeki önek (*prefix*) ve sonek'lerin (*suffix*) birleşiminden ön-son-ek 'lerden (*affix*) meydana gelen bir tablo oluşturup, her iki dildeki metinlerde sıkıştırma oranlarını kendi sezgisel (*heuristic*) algoritmalarını kullanarak incelemişlerdir (Önek ve sonek sıkıştırma tablosunun boyu 256 iken, İngilizce metinlerde başarı %45 , İbranice metinlerde ise %37 olmuştur.).

M.Burrows ve D.Wheeler (1983), içerisinde kendi yöntemi ile beraber diğer birkaç yöntemide kullanarak (RLE, MTF ve ARI), PKZIP ile aynı oranlarda sıkıştırma yapabilen ve hatta bazı dosyalarda daha iyi sonuç verebilen bir algoritma geliştirmişlerdir.

Faller (1973) ve Galleger (1978) yıllarında birbirlerinden habersiz olarak düşündükleri ve daha sonra Knuth (1985) tarafından geliştirilen, statik Huffman yönteminin dinamik uyarlaması ile kodlama ağacını, gelen veriye göre ağaç içerisinde anında güncellemektedirler. Bu algoritma **FGK** algoritması olarak da bilinmektedir.

Miller ve Wegman (1985) tarafından geliştirilen **LZMW** algoritmasının iki prensibinden biri sözlük dolduğunda sözlükten en az sıklıkta kullanılan kelimeyi silmek ve diğeri sözlüğe eklenen herbir kelimeyi, kendisinden önce gelmiş olan kelime ile birleştirerek sözlüğe eklemektir.

W.H.Hsu ve A.E.Zwarico'nun (1985) yayınladıkları makalede; metin dosyası, gerçel (*sanal*) görüntü, ikili (*binary*) bilgi, ses (*audio*) bilgisi ve hareketli (*animasyon*) görüntü gibi çeşitli tipteki verileri içeren karışık (*heterogeneous*) dosyalar için sıkıştırma yöntemi geliştirdiklerini belirtmişlerdir. Bu yöntemde metin dosyası içerisinde geçen her farklı bölüm için tek bir algoritma seçilmiş ve o metot uygulanmıştır. Bu çalışmada hem karışık (heterogeneous) dosyalar için sıkıştırma işlemi gerçekleştirilmiş hem de bir dosyanın sıkıştırılabilirliği ile ilgili olarak istatistiksel analiz metodu geliştirilmiştir.

Brent (1987) tarafından gerçekleştirilen **LZH** algoritması, Bloom (1996) tarafından gerçekleştirilen **LZP** algoritmalarıda bu konuda yapılan çalışmalardır.

Ross Williams (1991) tarafından gerçekleştirilen **LZRW** algoritmasındaki temel fikir aranan sözcüğün çırpı (*hash*) tablosu kullanarak bir adımda bulunmasıdır. Bu yöntem hızlı ama çok etkili değildir. Çünkü aranan sözcük herzaman en uzunu değildir.

R.Çölkesen (1995), yaptığı çalışmasında sözlük kullanan sıkıştırma algoritmaları için yazılım ve donanım önerileri getirmiş ve V.42bis sıkıştırma algoritmasını gerçek zamanda işleyebilen bir donanım tasarımını gerçekleştirmiştir.

A.Brodnik ve S.Carlsson (1998), yaptıkları çalışmada Huffman kodun geri çözülme zamanını azaltmışlardır. Kodun çözülme zamanı O(logN) den O(loglogN) düşürülmüştür.

4.2 Türk Dili ile İlgili Yapılan Çalışmalar

M.O. Kibaroğlu (1991), sondan eklemeli bir dil olan Türkçe için yazım kontrol programı geliştirmiştir.

H.L.Akın, S.Kuru, T.Güngör, İ.Hamzaoğlu ve D.Arbatlı 'nın (1993), çalışmalarında kök ve ek'leri kullanarak biçimbilimsel tabanlı yazım kontrolü ve hata düzelticisi yapmışlardır.

İ.Hamzaoğlu ve S.Kuru (1993) yaptıkları çalışmada, Türkçe'yi Türkçe'nin lehçelerinden birine (bu çalışmada Azeri Türkçe'si seçilmiştir) çevirmeyi yapmışlar ve bu çalışmada MT(Machine Translation) teknolojisini kullanmışlardır.

D.Gökçay ve U.Halıcı (1993), yaptıkları çalışmada sinir ağlarını (*Neural Network*) kullanarak, Türkçe'deki kelimelerin biçimbilimsel analizini yapmışlardır.

Z.Güngördü ve K.Oflazer (1993), yaptıkları çalışmada Türkçe için bir sözcüksel-işlevsel gramer geliştirmiş olup, yapı olarak basit ve girişik olan düz Türkçe cümleler ele alınıp başarı ile çözümlenmiştir.

Ç.Demir ve K.Oflazer (1993), yaptıkları çalışmada Türkçe için bir ATN* grameri geliştirip, Türkçe'nin basit ve girişik cümlelerini kapsayan bir alt kümesi için gerçekleştirmişlerdir.

İ.Kuruöz ve K.Oflazer (1994), yaptıkları çalışmada, Türkçe'nin tam kapsamlı iki aşamalı biçimbilimsel tanımlamasına dayanılarak geliştirilen bir sözcük türü işaretleyicisi sunmuşlardır.

T.Güngör (1995) , yaptığı çalışmada Türkçe'nin biçimbilimsel (*morfolojik*) yapısını ATN formunda inceleyip, biçimbilimsel analizi birbiriyle bağlantılı iki kısıma ayırıp morfotaktik** ve morfofonemik*** olarak ele alıp değerlendirmiştir. Kısacası Türkçe'nin biçimbilimsel yapısını açığa çıkarıp, bilgisayardaki gösterilimini oluşturmuştur.

D.Z.Hakkani, K.Oflazer ve İ.Çiçekli (1996), yaptıkları çalışmada serbest öğe sıralı bir dil olan Türkçe için yüzeysel üretici tasarımı gerçeklemişlerdir.

G.Tür ve K.Oflazer (1996), yaptıkları çalışmada, metinlerden bağımsız olarak elle oluşturulmuş, öğrenilmiş kuralları ve birikleştirilecek metinden elde edilen ek istatistiksel

---

* Augmented Transition Network (Genişletilmiş Geçiş Ağı)
** Eklerin, köke eklenme sırasının belirlenmesi
*** Eklerin, köke eklenirken gösterdikleri değişim

bilgileri kullanarak biçimbilimsel birikleştirme işlevini gerçekleştiren bir sistem geliştirmişlerdir.

5. TÜRKÇE METİNLERİN KAYIPSIZ SIKIŞTIRILMASI

Bu doktora çalışmasının amacı, bir Türkçe metin dosyasının içinde geçen kelimelerin, Türk dili kurallarına uygun olarak kök-gövde ve eklerine ayrılıp, geliştirilen kayıpsız sıkıştırma yöntemiyle sıkıştırılmasıdır. Bu çalışmada Huffman kodlama ağacı temel alınmış olup, yapılan deneysel incelemelerde Türkçe metin dosyalarının sıkıştırılmasında içeriğine bağlı olarak %50 ile %65 arasında verim elde edilmiştir. Geliştirilen yöntemin işleyişine ait ayrıntılar alt başlıklar halinde aşağıda verilmiştir.

Geliştirdiğimiz sıkıştırma algoritmalarında, veri sıkıştırma tekniklerinin test edilmesinde kullanılan Galgary Corpus ve Catenbury Corpus'a uygun olarak 14 adet Türkçe metinden "test-kümesi" oluşturulmuştur. Bu *test kümesi* içerisinde konuşma dili ile yazı diline en uygun yazı türü olan günlük gazetelerin köşe yazıları, yine konuşma diline yatkın yazı türü kullanan yazarlardan roman örnekleri, bir adet teknik çeviri ve bu doktora tezi örnek olarak kullanılmıştır (Çizelge 5.1).

Çizelge 5.1 Test kümesini oluşturan metin dosyaları

Sıra No	Dosya adı	Metnin gerçek uzunluğu (byte)	Toplam kelime sayısı	Toplam harf sayısı	Türü
1	Donat.txt	24.624	2.576	15.372	Makale
2	Gciva.txt	36.596	4.055	25.734	Makale
3	Takyol	41.377	4.976	31.417	Makale
4	Maşık.txt	48.881	5.716	36.732	Makale
5	Tubitak.txt	93.884	12.134	77.603	Teknik Çeviri
6	Phdthesis.txt	135.361	16.459	110.854	Tez
7	Ykemal.txt	160.240	21.337	126.808	Roman
8	Furuzan.txt	236.705	29.781	193.294	Roman
9	Omersey.txt	251.606	32.248	197.127	Roman
10	Oeksi.txt	666.112	65.965	408.887	Makale
11	Heper.txt	731.929	63.198	389.994	Makale
12	Ecolasan.txt	1.055.248	100.486	623.054	Makale
13	Mabirand.txt	1.795.226	182.255	1.157.298	Makale
14	Caltan.txt	2.147.084	230.755	1.458.870	Makale

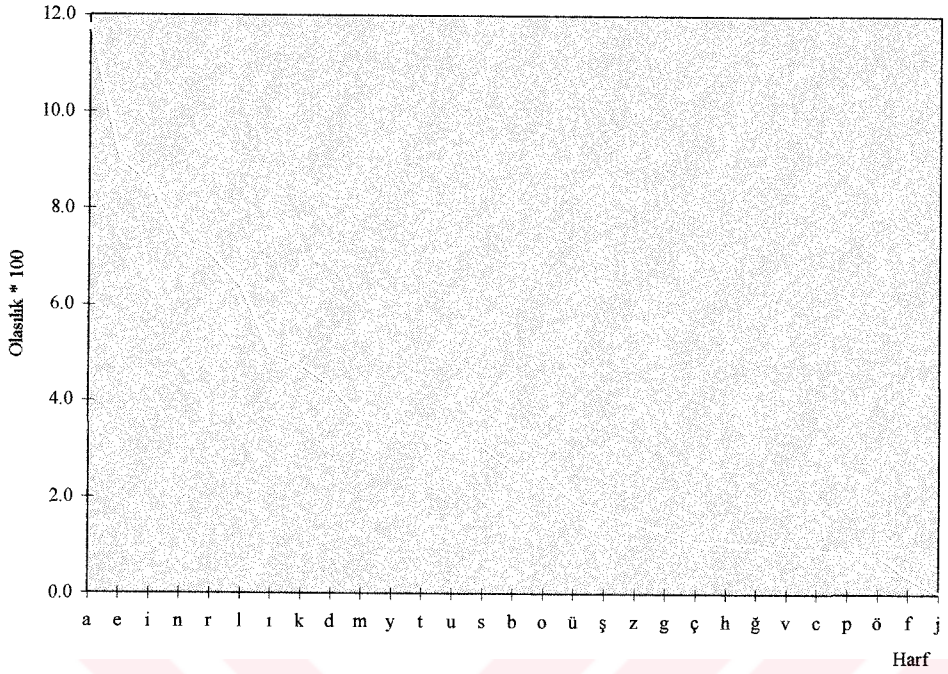
Bu çalışmada Karaktere Dayalı Statik Huffman:KrDSH; Karaktere Dayalı Dinamik Huffman:KrDDH; Heceye Dayalı Statik Huffman:HeDSH; İlk Heceye Dayalı Statik Huffman:IHeDSH; Köke Dayalı Statik Huffman:KökDSH; ve Kelimeye Dayalı Dinamik Huffman:KeDDH olarak kısaltılıp kullanılmıştır.

5.1 Türkçe'de Kullanılan Karakterlerin Kullanım Sıklıkları

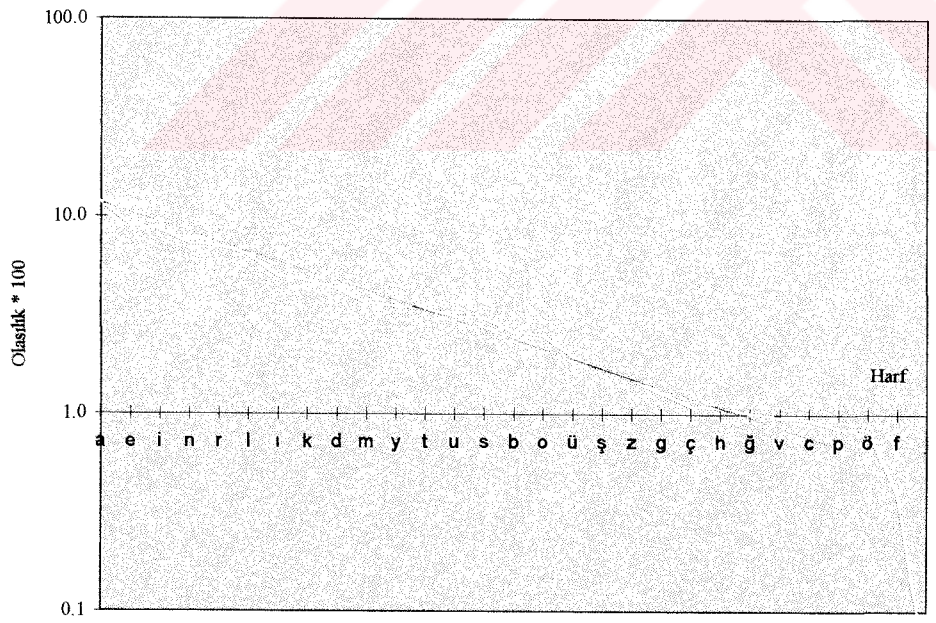
Bu bölümde ilk olarak Türkçe'de kullanılan karakterlerin kullanım sıklıkları incelenmiştir. Sıklık oranlarını elde etmek için test-kümesi'den yararlanılmıştır. Test-kümesi içerisinde geçen karakterlerin kullanım sıklıkları incelendiğinde Çizelge 5.2'deki değerler elde edilmiştir. Bu çizelgede Türkçe karaktelerin analizi yapılmış olup, ASCII tablosundaki diğer karakterler bu incelemenin dışında tutulmuştur. Çizelge 5.2'deki sıklık değerlerinden Türkçe harflerin kodlanması için gerekli ortalama bit uzunluğu, entropy hesabı yapılarak 4,37 bit bulunmuştur. Bu harflerin kullanım sıklıkları Şekil 5.1 ve Şekil 5.2'de grafik olarak gösterilmiştir.

Çizelge 5.2 Test-kümesi içerisinde geçen Türkçe harflerin kullanım sıklık ve olasılıkları

Harf	Sıklık	Olasılık	Harf	Sıklık	Olasılık	Harf	Sıklık	Olasılık
a	559.830	0,118	y	166.637	0,035	ç	52.430	0,011
e	429.104	0,091	t	162.316	0,034	h	51.108	0,011
i	404.978	0,085	u	148.754	0,031	ğ	48.007	0,010
n	347.395	0,073	s	146.462	0,031	v	45.875	0,010
r	335.494	0,071	b	123.324	0,026	c	44.477	0,009
l	303.604	0,064	o	120.011	0,025	p	40.442	0,009
ı	237.281	0,050	ü	93.238	0,020	ö	39.916	0,008
k	226.175	0,048	ş	80.801	0,017	f	18.988	0,004
d	205.241	0,043	z	71.982	0,015	j	2.453	0,001
m	174.081	0,037	g	61.013	0,013			



Şekil 5.1 Türk alfabesindeki karakterlerin kullanım sıklıkları eğrisi



Şekil 5.2 Türk alfabesindeki karakterlerin kullanım sıklıkları eğrisi (yarı logaritmik eksen)

Şekil 5.2'deki Türk alfabesini oluşturan karakterlerin kullanım sıklığı eğrisi yarı logaritmik eksen sisteminde çizilmiştir. Bu eğrinin değişimi bir lineer doğru denklemine yakın değer göstermekte olup, $y = a * e^{-bx}$ fonksiyonu ile gösterilir. Bu fonksiyonun belirgin hale gelmesi **a** ve **b** katsayılarının bulunması ile mümkündür. $y = a * e^{-bx}$ üstel fonksiyonunun iki tarafının da logaritması alınır ve gerekli dönüşümler yapılırsa:

$$\ln y = \ln(a) + (-bx)$$

$$\downarrow \quad \downarrow \quad \downarrow$$

$$Y = A + Bx$$

doğru denklemi elde edilir. Doğru denkleminin **A** ve **B** katsayılarını bulmak için *En Küçük Kareler* yöntemi kullanılarak,

$$\begin{bmatrix} n & \sum x \\ \sum x & \sum x^2 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} \sum \ln y \\ \sum x \ln y \end{bmatrix}$$

matris formu elde edilir. Bu form lineer denklem sistemlerinin, $[M][X] = [N]$ şeklindeki genel ifadesine benzetilirse, $[M]$ matrisinin yerini $\begin{bmatrix} n & \sum x \\ \sum x & \sum x^2 \end{bmatrix}$ matrisi, (N) vektörünün yerini de **x** ve buna bağlı ölçülen **y** değerleri ile hesaplanan, $\begin{bmatrix} \sum \ln y \\ \sum x \ln y \end{bmatrix}$ matrisi alır. Bu sistemin bilinmeyenler vektörünü ise $\begin{bmatrix} A \\ B \end{bmatrix}$ elemanları oluşturur.

Yapılan uygulamada n = 29 için,

$\sum x = 435$, $\sum x^2 = 8555$, $\sum \ln y = 335$, $\sum x \ln y = 4780$ olarak hesaplandığında, A ve B katsayılarının değerleri, A=13,35, B= -0,12 olup,

a ve b'nin değeri, $a = e^{13.35} = 627.814$ $b \cong B = 0,12$ olarak bulunup, doğru denklemi, $y = 627.814e^{-0,12x}$ şeklinde elde edilir.

Bu değer Zipf kanunlarıyla verilen ifadelerdeki logaritmik yapıya uygun düşmektedir. Ancak $y = 627.814e^{-0,12x}$ ifadesinde en çok kullanılan birinci karakter için,

x = 1 olup,

$y_1 = 627.814e^{-0,12x} = 627.814 * 0,88 = 556.821$ değeri elde edilmiştir.

Gerçek değer 559.830 olup, mutlak hata |556.821 - 559.830| = 3.009

Bağıl hata ise 0,005 iken,

x =13 için,

$y_{13} = 627.814e^{-0.12x} = 627.814 * 0,21 = 131.926$ olup,

Gerçek değer 148.754 olup, mutlak hata |131.926 - 148.754| = 16.828

Bağıl hata ise 0,12 olmaktadır.

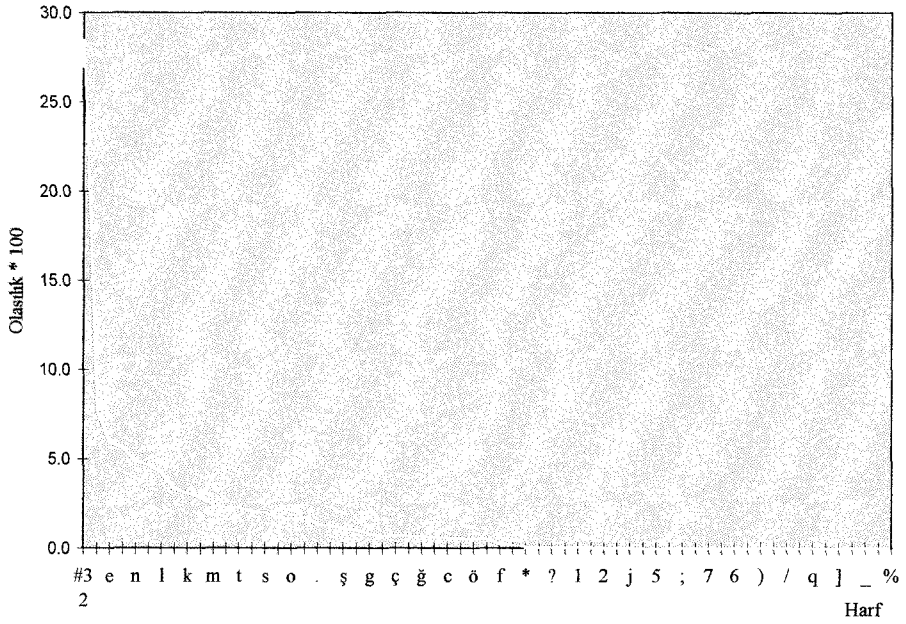
Test-kümesi içerisinde geçen bütün karakterlerin kullanım sıklık ve olasılıkları Çizelge 5.3' de verilmekte olup, karakterlerin kodlanması için gerekli ortalama bit uzunluğu, entropy hesabı yapılarak 4,18 bit olarak bulunmuştur. Şekil 5.3'de de bu karakterlerin kullanım sıklıkları grafiksel olarak gösterilmiştir.

Türkçe bir metinde, kullanılan karakterlere göre Huffman ağacı oluşturularak yapılan sıkıştırma algoritmasında değişik karakter sayısı (alfabetik, nümerik, özel işaretler) t=63 tane olup, düğüm noktası sayısı d=2*t-2 bağıntısıyla d=2*63-2=124 olarak bulunur.

Metin içerisinde kullanılan karakterlerin, Zipf fonksiyonuna göre bir dağılım göstermesi halinde, Huffman ağacında *l* seviyeyi, t de kullanılan farklı karakter sayısını göstermek üzere, N elemanlı bir metinde sadece t değişik karakter tekrar ediyor ise kodlanmış karakterin maksimum bit uzunluğu *l* olmaktadır.

Çizelge 5.3 Test-kümesi içerisinde kullanılan bütün karakterler

Karakter	Sıklık	Olasılık	Karakter	Sıklık	Olasılık
#32	2.040.374	0,285	f	18.988	0,003
a	559.830	0,078	“	14.491	0,002
e	429.104	0,060	*	8.337	0,001
i	404.978	0,057	-	6.821	0,001
n	347.395	0,049	?	6.214	0,0009
r	335.494	0,047	:	4.128	0,0006
l	303.604	0,042	1	4.261	0,0006
ı	237.281	0,033	0	3.977	0,0006
k	226.175	0,032	2	2.830	0,0004
d	205.241	0,029	9	2.740	0,0004
m	174.081	0,024	j	2.453	0,00034
y	166.637	0,023	!	2.185	0,00031
t	162.316	0,023	5	1.785	0,00025
u	148.754	0,021	3	1.569	0,00022
s	146.462	0,020	;	1.553	0,00022
b	123.324	0,017	8	1.376	0,00019
o	120.011	0,017	7	1.216	0,00017
#13 #10	125.012	0,017	4	1.066	0,00015
.	96.138	0,013	6	916	0,00013
ü	93.238	0,013	(	886	0,00012
ş	80.801	0,011	)	886	0,00012
z	71.982	0,010	w	664	0,00009
g	61.013	0,009	/	95	0,00001
,	53.864	0,008	x	61	0,000009
ç	52.430	0,007	q	29	4E-6
h	51.108	0,007	+	10	1E-6
ğ	48.007	0,007	]	8	1E-6
v	45.875	0,006	[	8	1E-6
c	44.477	0,006		6	1E-6
p	40.442	0,006	=	5	1E-6
ö	39.916	0,006	%	1	1E-7
‘	39.502	0,006			



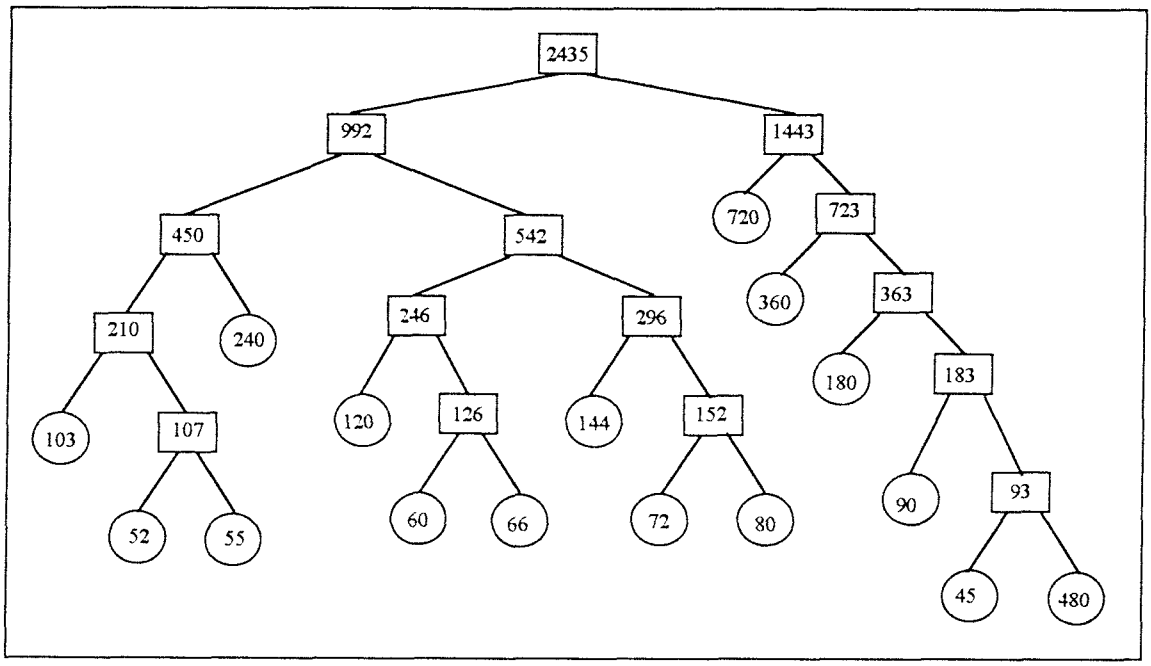
Şekil 5.3 Test-kümesi'ndeki tüm karakterlerin kullanım sıklık eğrisi

Zipf kanununa uygun dağılım gösteren t=16 değişik karakterin kullanıldığı n_1=720 elemanlı bir metin için elde edilen dağılımlar Şekil 5.4'de, Huffman ağacı da Şekil 5.5'de gösterilmiştir. Bu ağacın seviyesi l = 6 olup, sıkıştırılmış metinde her karakter için ortalama bit uzunluğu, S=3,25 sıkıştırma oranı, ρ = 40,61 sıkıştırma verimi de η = 59,39 olarak hesaplanmıştır.

720	360	240	180	144	120	103	90	80	72	66	60	55	52	48	45

Şekil 5.4 16 Adet karakterin metin içerisindeki kullanım sıklıkları*

* n_1=720 alınıp diğer elemanların değeri Zipf kanununa göre hesaplanmıştır.



Şekil 5.5 Örnek Huffman ağacı

5.2 Türkçe Bir Metnin Huffman Algoritması Temel Alınarak Sıkıştırılması

Türkçe metinlerin sıkıştırılmasında yedi farklı yöntem kullanılmıştır. Bunlar sırası ile *Karaktere Dayalı Statik Huffman* (KrDSH) algoritması, *Karaktere Dayalı Dinamik Huffman* (KrDDH) algoritması, *Hecelere Dayalı Statik Huffman* (HeDSH) algoritması, *İlk Heceye Dayalı Statik Huffman* (IHeDSH) algoritması, *Köke Dayalı Statik Huffman* (KökDSH) algoritması, *Kelimeye Dayalı Dinamik Huffman* (KeDDH) algoritması ve *Geliştirilmiş Kelimeye Dayalı Dinamik Huffman* (GKeDDH) algoritmasıdır. Bu yöntemler *test-kümesi* içerisindeki dosyalara uygulanmış ve performans değerleri elde edilmiştir. Yukarıda adı geçen yedi farklı yöntemde kodlanacak giriş bilgisi farklı olup, herbiri için kod oluşturulurken Huffman kodlama ağacından yararlanılmıştır.

5.2.1 Huffman algoritmasına genel bakış

N karakterden oluşan bir metin içerisinde t adet farklı (*karakter/gövde-kök/ek-hece*) tipteki sembollerin kullanım sıklıkları sırasıyla $n_1,n_2,n_3,...,n_t$ olarak verilmiş olsun (2.13). Huffman kodlamasında kullanılacak olan bu sıklık bilgilerini küçükten büyüğe doğru sıralanmış bir dizi olarak düşünelim. Bu dizi içerisinde bulunan $n_1,n_2,\ldots,n_t$ gibi sıklık bilgileri, kuracağımız kodlama ağacının yapraklarını oluşturacaktır. Yazılmış olan sıkıştırma algoritmasında, sıklık bilgilerini sıralarken *Quick Sort* algoritması kullanılmış olup, karşılaştırma karmaşıklığı en kötü durumda,

$$O(t \log_2 t) \qquad (5.1)$$

dir. Aşağıdaki anlatımda,

S : Kullanım sıklık dizisi

K : Sembol dizisi

I : İndis dizisi

olarak kullanılmaktadır.

K : (bayrak,de,ben,sen,git,okul,gün,pazar,sınav)

S : (1,2,5,11,17,30,40,45,50)

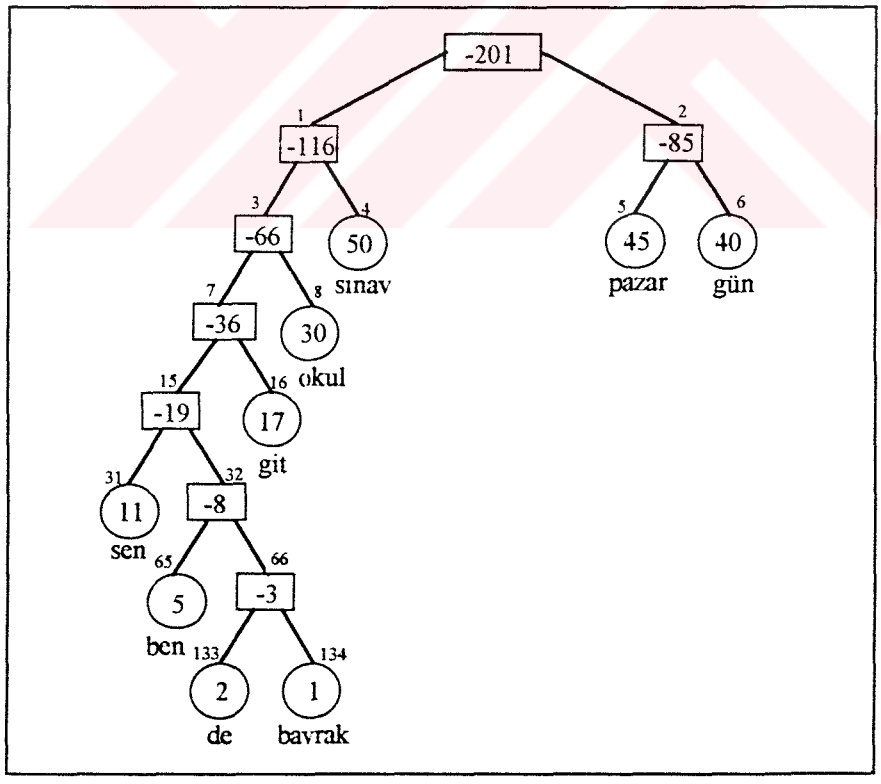
verilmiş olup **t=9** dur. Huffman ağacı oluşturulurken "S" dizisinin ilk iki elemanının toplamı $[(S_i+S_{i+1}), i=1,...,(2t-2)]$ sıralamayı bozmayacak şekilde "S" dizisine eklenir. Eğer S_i ve S_{i+1} değerleri, ağacın (j) ve (j+1)'inci düğümündeki yaprakları oluşturuyorsa (j/2) değeri de bu yaprakların ebeveyn'ini (*parent*) oluşturur. Bu işleme S_{i+2} ve S_{i+3}'üncü elemanlarının toplamının diziye eklenmesiyle devam edilir. "S" dizisinin eleman sayısı (2*t-2) ye ulaşınca işleme son verilir. "S" dizisine her yeni eleman negatif işaretli olarak eklenmektedir (toplama işlemi her elemanın mutlak değeri alınarak gerçekleştirilir). Bu negatif değerler Huffman ağacının ara kök değerleri olup, herhangi bir sembolün kullanım sıklığı değeri değildir. Bir başka deyişle ağacın yaprak değerlerinden biri değildir. Yapılan işlem sonucunda "S" dizisinin son hali,

S : (1,2,-3,5,-8,11,17,-19,30,-36,40,45,50,-66,-85,-116) şeklini alır.

Yukarıda bahsedilen işlemin karşılaştırma karmaşıklığı (her toplamın dizi içerisinde sıralamayı bozmadan yerleştirilebilmesi için yapılan arama ve değerin dizinin ilgili gözüne yerleştirilmesi için dizinin kaydırılması) **t** eleman için,

$$O\left(\frac{t^2}{2}\right) \tag{5.2}$$

olarak verebiliriz. "S" dizisini ağaca yerleştirmek için bir ara işleme gerek duyulmaktadır. "S" dizisi tersten işlem görecek şekilde yerleştirilir, S[0]=-116, S[1]=-85,...,S[15]=1 "I" indis dizisinin ilk iki elemanı da I[0]=1 ve I[1]=2 değerini alır. Bu, (-116) değerinin ağacın 1 numaralı düğümünde, (-85) değerinin de ağacın 2 numaralı düğümünde bulunduğunu göstermektedir (Şekil 5.6).



Şekil 5.6 Sıklık bilgilerinin Huffman ağacına yerleştirilmesi

Daha sonra işlem "S" dizisinin ilk negatif değere sahip elemanından başlar. "I" dizisinin birinci elemanındaki değerin iki katının bir fazlasını "I" dizisinin ilk boş gözüne, iki fazlasını da bir sonraki gözüne yerleştirip, "S" dizisinin bir sonraki elemanına geçilir. $S_{i+1} < 0$ ise aynı işleme devam edilir. Eğer $S_{i+1} > 0$ ise, S_{i+1} değeri bir yaprak değeri olup I[w]'deki (w=0,...,2t-2) değer S_{i+1}'in ağaçtaki düğümünü göstermektedir. Bu işlem "I" dizisinin eleman sayısı (2*t-2) oluncaya kadar devam ettirilir. "S" ve "I" dizilerinin son durumu Çizelge 5.4'deki gibi olur.

Çizelge 5.4 S ve I dizisinin son hali

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	-116	-85	-66	50	45	40	-36	30	-19	17	11	-8	5	-3	2	1
I	1	2	3	4	5	6	7	8	15	16	31	32	65	66	133	134

Huffman ağacının yapraklarında yer alan her sembolün Huffman kodlarını oluştururken "I" dizisinden yararlanılmaktadır. "S" dizinin her pozitif değerine ($S_i > 0$) karşılık düşen, "I" dizisinin aynı indisindeki değer o sembolün Huffman kodunu vermektedir.

Huffman ağacının ve her bir sembolün kodlarının oluşturulabilmesi için gerekli kod parçası aşağıda verilmiştir. Bu işlemin maksimum karmaşıklığı, (5.3)'deki gibidir.

$$O\left((3*t-2)+t*\left(\log_2 * t\right)\right) \qquad (5.3)$$

İleriki bölümlerde anlatılacak yöntemlerden alınan sonuçlar (sıkıştırma verimi η, sıkıştırma süresi SS ve genişletme süresi GT) 100Mhz. Pentium işlemci ve ortalama erişim hızı 16ms. olan sabit diske sahip bir bilgisayardan alınmıştır. Yazılmış olan bütün algoritmalar Win95 işletim sistemi altında çalışan Borland Pascal 7.0'da yazılmıştır.

Huffman ağacını ve sembollerinin kodlarını oluşturan algoritma:

```
kk := t-1;
I[0] := 1; I[1] := 2;
 j := 2;
 w := 0;

for  x := (2*t-2)-1  downto 0 do
begin
  if  S[x]  <  0  then
  begin
     m := I[w];
     I[j] := 2*m+1;
     I[j+1] := 2*m+2;
     j := j+2;
     w := w+1;
  end
  else
   begin
        point :=  I[w];
        w := w + 1;
        y := 0;
        sembol := S[kk];

    while  point  >  0 do
    begin
       if  point  mod 2  =  0 then
       begin
          code := code or 1;
          point := (point-2) shr 1;
       end
       else  point := (point-1) shr 1;
       code :=code shl 1;
       y := y + 1;
    end;
    code := code  shr 1;
    code_length := y;
   kk := kk-1;
    end;
end;
```

5.3 Türkçe Bir Metnin Karaktere Dayalı Sıkıştırılması

Türkçe metinlerin karaktere dayalı sıkıştırılmasında iki farklı yöntem kullanılmıştır. Bu iki yöntem *Karaktere Dayalı Statik Huffman* algoritması ile *Karaktere Dayalı Dinamik Huffman* algoritmasıdır. Bütün yöntemlerden elde edilecek olan performans sınırlarını belirlemek amacıyla her yöntem, *test-kümesi* içerisindeki dosyalara ayrı ayrı uygulanmıştır.

5.3.1 Karaktere Dayalı Statik Huffman yöntemi

Karaktere Dayalı Statik Huffman algoritmasında, Türkçe'deki karakterlerin kullanım sıklıklarının önceden belirlenerek, statik bir şablonun oluşturulması gerekmektedir.

5.3.1.1 Karaktere dayalı statik şablonun oluşturulması

Karaktere Dayalı Statik Huffman algoritmasında kullanılacak olan ve "stksab" ismini verdiğimiz şablonu oluşturmak için sırası ile aşağıdaki işlemler yapılır.

i) *Test-kümesi* içerisindeki dosyalar taranarak kullanılmış olan bütün karakterlerin kullanım sıklık bilgileri çıkarılır.
ii) Metin dosyaları içerisinde yer almayan fakat ileride bir başka metin dosyası içerisinde çıkabilme ihtimali olan karakterlere minimum bir sıklık değeri atanır.
iii) Bölüm 5.2.1'de anlatıldığı üzere her bir karakterin Huffman kod karşılığı üretilir.

Bu adımlardan sonra elimizdeki "stksab" dosyasının içerisinde **107** adet farklı karakterin kendisi, Huffman kodları (decimal olarak değeri) ve kodun bit uzunluğu bilgisi tutulmaktadır. Karaktere dayalı statik şablon, sıkıştırma ve genişletme modüllerinde yer alan aramaları hızlandırmak için Huffman kod değerlerine göre sıralı olarak oluşturulmuştur.

Karaktere dayalı statik şablonda yer alan karakterler, kullanım sıklıkları ve olasılıkları EK 4'de verilmiştir.

5.3.1.2 KrDSH yöntemi ile Türkçe bir metnin sıkıştırılması/genişletilmesi

Hazırlanmış olan karaktere dayalı statik şablon kullanılarak (statik şablonun belleğe çekilmesi 60ms sürmektedir), sıkıştırılmak istenen metin dosyası içerisindeki karakterlerin Huffman kodları 16 bitlik (*word*) gruplar halinde bloklanıp, sıkıştırma işlemi gerçekleştirilmektedir. Sıkıştırılmış dosyanın ilk iki byte'ı, genişletme modülünde kullanılacak olan artık bit sayısını* ve şablon içerisindeki toplam karakter sayısını tutmaktadır. Genişletme bölümünde, ilgili Huffman kodlarının statik şablon içerinde aranması için İkili Arama (*Binary Search*) algoritması kullanılmştır. İkili arama algoritmasında **t** farklı eleman için maksimum karmaşıklık,

$$O(t * \log_2 t) \qquad (5.4)$$

olarak hesaplanır. *Karaktere Dayalı Statik Huffman* yöntemi, *test-kümesine* uygulandığında her dosyanın sıkıştırma verimi, sıkıştırma ve genişletme süreleri gerçek metin dosyaları ile karşılaştırmalı olarak Çizelge 5.5'de, performans sınırlarını gösteren grafik de Şekil 5.7'de verilmektedir.

Çizelge 5.5'den *Karaktere Dayalı Statik Huffman* yöntemi kullanılarak bir metin dosyasının sıkıştırılmasında, ortalama **%45** 'lik sıkıştırma verimi elde edilmiş olduğu gözlenmektedir. Bu ve ileride bahsedilecek olan diğer yöntemlerin, ortalama sıkıştırma veriminin hesabında (5.5)'deki ağırlıklı ortalama formülünden yararlanılmıştır (l: sıkıştırılmış dosya boyu, k: dosya adedi, η: sıkıştırma verimi).

* Sıkıştırılmış dosyanın en son byte'ındaki kaç bitin gerçek bilgi olmadığını gösterir.

$$\frac{\sum_{i=1}^{k} l_i \eta_i}{\sum_{i=1}^{k} l_i} \qquad (5.5)$$

Karaktere Dayalı Statik Huffman yönteminde, büyük metinlerdeki başarı, küçük metinlere oranla daha iyidir. Genişletme süresi de, sıkıştırma süresinin yaklaşık **5,7** katı daha fazla sürmektedir (GS = 5,7 * SS).

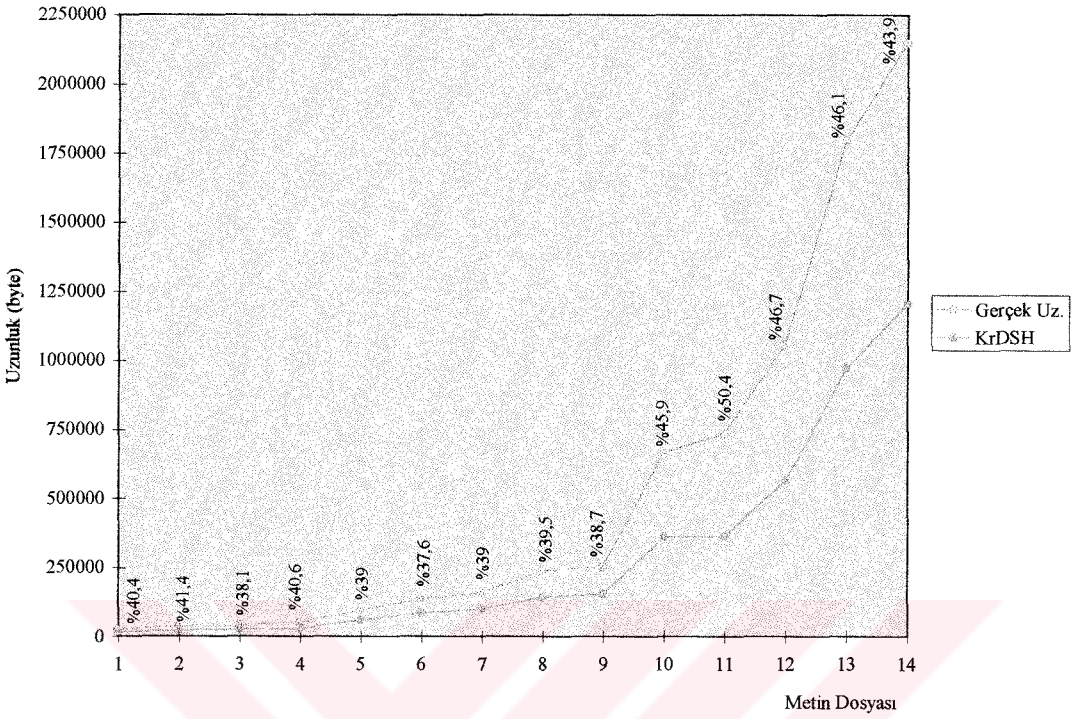
Çizelge 5.5 Gerçek metin dosyası ile KrDSH' dan alınan sonuçların karşılaştırılması

Sıra No	Dosya Adı	Gerçek Uzunluk	KrDSH	SS (ms)	η %	GS (ms)
1	Donat.txt	24.624	14.684	170	40,4	720
2	Gciva.txt	36.596	21.446	220	41,4	1.050
3	Takyol.txt	41.377	25.604	220	38,1	1.270
4	Masik.txt	48.881	29.038	220	40,6	1.430
5	Tubitak.txt	93.884	57.254	490	39,0	2.850
6	Phdthesis.txt	135.361	84.416	660	37,6	4.023
7	Ykemal.txt	160.240	97.736	820	39,0	4.890
8	Furuzan.txt	236.705	143.142	1.160	39,5	7.200
9	Omersey.txt	251.606	154.282	1.210	38,7	7.750
10	Oeksi.txt	666.120	360.174	3.020	45,9	18.130
11	Heper.txt	731.929	363.148	3.020	50,4	18.240
12	Ecolasan.txt	1.055.244	562.130	4.670	46,7	28.340
13	Mabirand.txt	1.795.226	968.302	8.130	46,1	48.880
14	Caltan.txt	2.147.095	1.204.846	10.000	43,9	60.800

5.3.2 Karaktere Dayalı Dinamik Huffman yöntemi

Karaktere Dayalı Dinamik Huffman yönteminde, karakterlerin kullanım sıklıklarının önceden bilinmesine gerek yoktur. Karakterlerin kullanım sıklıkları, sıkıştırılacak olan metnin içeriğine bağlı olarak değişir. Bir metin dosyası içerisinde herhangi bir karaktere

karşılık gelen kod, diğer bir metin dosyası içerisinde aynı karaktere farklı bir kodun karşı gelmesine neden olur.



Şekil 5.7 Gerçek metin dosyası ile KrDSH yönteminin karşılaştırılması

5.3.2.1 KrDDH yöntemi ile Türkçe bir metnin sıkıştırılması/genişletilmesi

Karaktere Dayalı Dinamik Huffman yönteminin, *Karaktere Dayalı Statik Huffman* yönteminden farkı yukarıda belirtildiği gibi metin içerisindeki karakterlerin kullanım sıklıklarının çıkarılması, Huffman ağacının oluşturulması ve her bir karakter için ilgili Huffman kodlarının elde edilmesi ayrı bir program olarak değil de, sıkıştırma modülünün içerisinde gerçekleştirilmiş olmasıdır. Sıkıştırma işlemi bu aşamadan sonra gerçekleştirilir. Her iki algoritmada da sıklıkların sıralanması, Huffman ağacının oluşturulması, kodların elde edilmesi ve genişletme modülündeki arama işlemlerinin karmaşıklıkları aynıdır.

Sıkıştırma modülündeki işlem karmaşıklığı (fl = dosya boyu (byte), acl = ortalama kod uzunluğu) denklem (5.6)'daki gibi hesaplanır.

$$O(fl * acl) \tag{5.6}$$

Genişletme modülündeki işlem karmaşıklığı ise (*cl* = sıkıştırılmış dosya boyu, *t* = farklı karakter sayısı) denklem (5.7)'deki gibi hesaplanmaktadır.

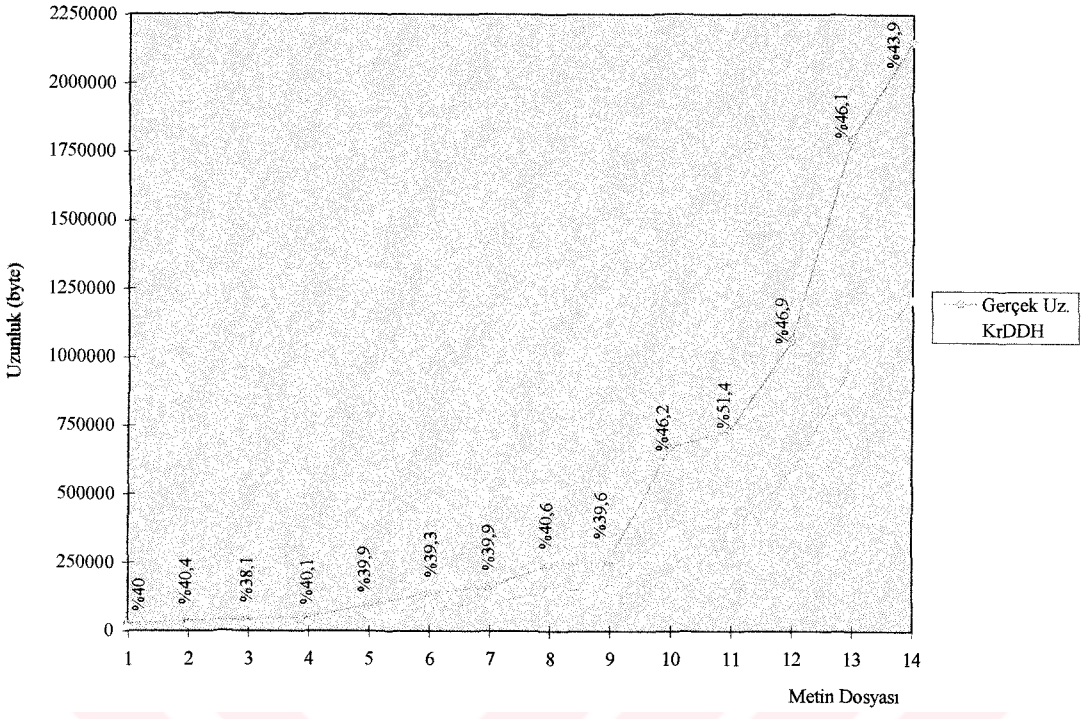
$$O\left(cl * 8 * \left(t * \log_2 t\right)\right) \tag{5.7}$$

Karaktere Dayalı Dinamik Huffman yöntemi, *test-kümesine* uygulandığında her dosyanın sıkıştırma verimi, sıkıştırma ve genişletme süreleri, gerçek metin dosyaları ile karşılaştırmalı olarak Çizelge 5.6'da, performans sınırlarını gösteren grafik de Şekil 5.8'de verilmektedir.

Çizelge 5.6'dan *Karaktere Dayalı Dinamik Huffman* yöntemi kullanılarak bir metin dosyasının sıkıştırılmasında, ortalama **%45,2** 'lik sıkıştırma verimi elde edilmiş olduğu gözlenmektedir.

Çizelge 5.6 Gerçek metin dosyası ile KrDDH' dan alınan sonuçların karşılaştırılması

Sıra No	Dosya Adı	Gerçek Uzunluk	KrDDH	SS(ms)	η %	GS (ms)
1	Donat.txt	24.624	14.769	170	40,0	660
2	Gciva.txt	36.596	21.793	220	40,4	1.050
3	Takyol.txt	41.377	25.601	220	38,1	1.210
4	Masik.txt	48.881	29.297	280	40,1	1.370
5	Tubitak.txt	93.884	56.456	490	39,9	2.690
6	Phdthesis.txt	135.361	82.145	720	39,3	4.000
7	Ykemal.txt	160.240	96.249	880	39,9	4.400
8	Furuzan.txt	236.705	140.653	1.260	40,6	6.600
9	Omersey.txt	251.606	151.999	1.370	39,6	7.090
10	Oeksi.txt	666.120	358.682	3.300	46,2	18.240
11	Heper.txt	731.929	355.987	3.360	51,4	17.240
12	Ecolasan.txt	1.055.244	560.838	5.280	46,9	26.920
13	Mabirand.txt	1.795.226	968.048	8.010	46,1	47.180
14	Caltan.txt	2.147.095	1.204.362	11.580	43,9	57.500



Şekil 5.8 Gerçek metin dosyası ile KrDDH yönteminin karşılaştırılması

Bu yöntemde de genişletme süresi, sıkıştırma süresinin yaklaşık **5,2** katı daha fazla sürmektedir (GS=5,2*SS).

5.3.3 KrDSH ile KrDDH yöntemlerinin karşılaştırılması

Karaktere Dayalı Statik Huffman ile *Karaktere Dayalı Dinamik Huffman*'dan alınan sonuçların birbirine çok yakın değerler olduğu Çizelge 5.7'deki sıkıştırma veriminden de anlaşılmaktadır. Yinede *Karaktere Dayalı Dinamik Huffman* yönteminin daha başarılı olduğu gözlenmektedir (Şekil 5.9). Bunun sebebi, dinamik Huffman kodlamasında karakterlerin kullanım sıklıklarının metine bağlı olarak değişmesidir. *Karaktere Dayalı Statik Huffman* yönteminde kullanılan statik şablonumuzda 107 karakter var iken, *Karaktere Dayalı Dinamik Huffman* yönteminde sıkıştırılacak her metin dosyası için karakter sıklıkları tarandığında daha az sayıda karakter elde edilebilir. Bu durum Huffman kodlama ağacının farklı oluşmasına neden olur. Bazı durumlarda bir karaktere karşı gelen kodun bit uzunluğu

kısalabilir. Bu da bize *Karaktere Dayalı Dinamik Huffman* yönteminden daha iyi sonuç alınabileceğini gösterir.

Çizelge 5.7 Gerçek metin dosyası ile KrDSH ve KrDDH yönteminden alınan sonuçlar

			KrDSH			KrDDH		
Sıra No	**Dosya Adı**	**Gerçek Uzunluk**	**η %**	**SS(ms)**	**SS/GS (ms)**	**η %**	**SS(ms)**	**SS/GS (ms)**
1	Donat.txt	24.624	40,4	170	0,24	40,0	170	0,26
2	Gciva.txt	36.596	41,4	220	0,21	40,4	220	0,21
3	Takyol.txt	41.377	38,1	220	0,17	38,1	220	0,18
4	Masik.txt	48.881	40,6	220	0,15	40,1	280	0,20
5	Tubitak.txt	93.884	39,0	490	0,17	39,9	490	0,18
6	Phdthesis.txt	135.361	37,6	660	0,16	39,3	720	0,18
7	Ykemal.txt	160.240	39,0	820	0,17	39,9	880	0,20
8	Furuzan.txt	236.705	39,5	1.160	0,16	40,6	1.260	0,19
9	Omersey.txt	251.606	38,7	1.210	0,16	39,6	1.370	0,19
10	Oeksi.txt	666.120	45,9	3.020	0,17	46,2	3.300	0,18
11	Heper.txt	731.929	50,4	3.020	0,17	51,4	3.360	0,19
12	Ecolasan.txt	1.055.244	46,7	4.670	0,16	46,9	5.280	0,20
13	Mabirand.txt	1.795.226	46,1	8.130	0,17	46,1	8.010	0,18
14	Caltan.txt	2.147.095	43,9	10.000	0,16	43,9	11.580	0,20

Karaktere Dayalı Statik Huffman ile *Karaktere Dayalı Dinamik Huffman* yöntemleri arasındaki en büyük fark, *Karaktere Dayalı Dinamik Huffman* yönteminde statik bir şablona ihtiyaç olmamasıdır. *Karaktere Dayalı Dinamik Huffman* yönteminde, kod çözücü tarafın karakterlerin hangi kodlar ile ifade edildiğini bilmesi gerekir. Bu sebeple *Karaktere Dayalı Dinamik Huffman* yönteminde sıkıştırılmış dosyanın önünde başlık (*header*) ismini vereceğimiz bir bölüm olmalıdır.

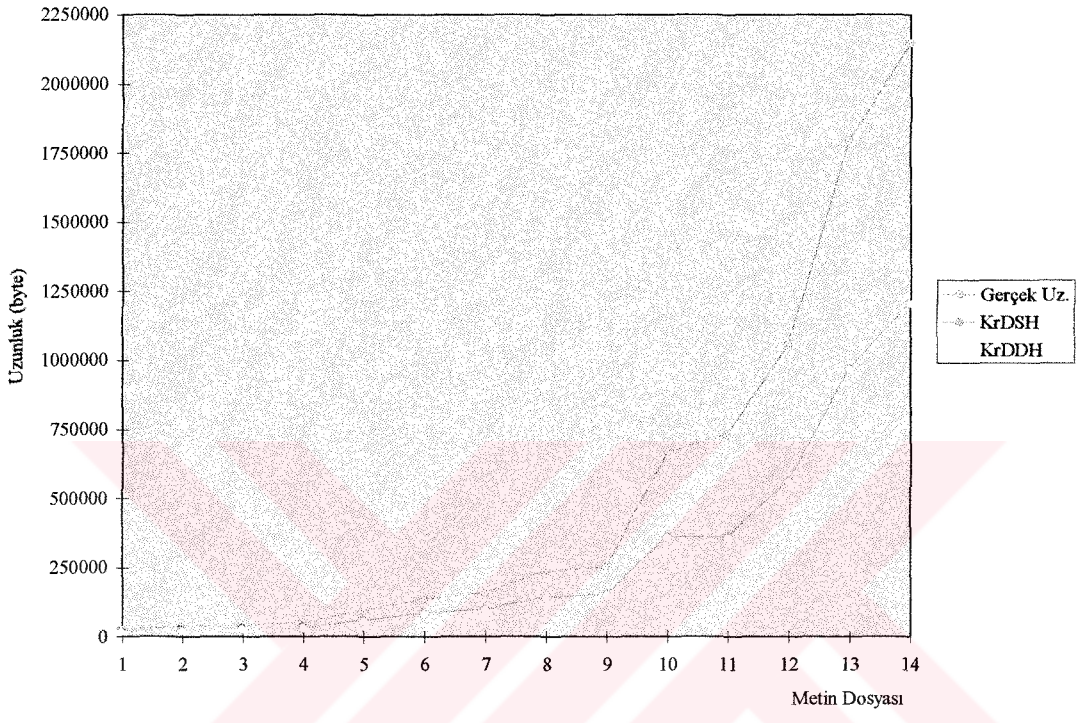
Başlığın uzunluğu,

cc : Kullanılan karakter sayısı

cf : Karakterin kullanım sıklığı (4 byte)

$cc * (1 + cf) = (cc * 5)$ byte olarak hesaplanır.

Çizelge 5.7'den görüldüğü gibi, *Karaktere Dayalı Statik Huffman* yönteminde genişletme süresi, sıkıştırma süresinden ortalama **5,7** kat daha fazla olmasına karşılık (GS = SS * 5,7), bu oran *Karaktere Dayalı Dinamik Huffman* yönteminde **5,2** kat daha fazladır (GS=SS*5,2).



Şekil 5.9 Gerçek metin dosyası ile KrDSH ve KrDDH yönteminin sıkıştırma verimlerinin karşılaştırılması

Karaktere Dayalı Statik Huffman yönteminin sıkıştırma süresi, *Karaktere Dayalı Dinamik Huffman* yöntemine göre daha kısadır. *Karaktere Dayalı Statik Huffman* yöntemi hazır şablon kullandığı için karakterlerin Huffman kodlarını oluşturmak için zaman harcamaz. Buna karşılık *Karaktere Dayalı Dinamik Huffman* yönteminin genişletme süresi, *Karaktere Dayalı Statik Huffman* yöntemine göre daha kısadır. *Karaktere Dayalı Statik Huffman* yöntemine göre daha az sayıda karakter ile işlem yaptığından arama zamanı azalmaktadır.

5.4 Karakter Sıkıştırmadan Sembol Sıkıştırmaya Geçiş

Türkçe metinlerin, biçimbilimsel (*morfolojik*) incelenmesi ve buna bağlı olarak metinlerde geçen kelimelerin hece,kök ve eklerine ayrılıp kodlanmasının, karaktere dayalı sıkıştırma yöntemine göre daha iyi performans vereceği düşünülerek Türkçe'nin biçimbilim yapısından yararlanılmasına karar verilmiştir. Biçimbilim, dilleri kelime yapıları bakımından inceler. Bu incelemede diller üç ana gruba ayrılır:

- Yalınlayan diller
- Bükümlü diller
- Bitişken (*bağlantılı*) diller

Yalınlayan diller, tek heceli diller olup, kelimeler ek almadıkları gibi değişikliğe de uğramazlar. Çince ve bazı Afrika dillerini bu gruba örnek olarak gösterebiliriz.

Bükümlü diller, yeryüzünün en yaygın ve gelişmiş dillerinden olup, kelime üretmede kökler de değişikliğe uğrar. Arap dilleri ve Hint Avrupa dilleri bu sınıfa girer.

Bitişken dillerde (*agglutinative*), kelimenin köklerine çeşitli ekler takılarak, kelimeye yeni anlamlar yüklenebildiği gibi, yeni kelimeler de elde edilebilir. Türkçe, Altay dilleri bazı Asya ve Afrika dilleri bu sınıf dillere örnek olarak gösterilir.

Bu bölümde üç farklı yöntemden bahsedilmektedir. İlk olarak bir kelimenin ayrılabilir parçaları olarak heceler düşünülmüş ve Türkçe metin dosyası içerisindeki kelimeler hecelerine ayrılarak sıkıştırma yoluna gidilmiştir. Daha sonra Türkçe'nin bitişken dil olması, kelimelerin kök ve eklerine ayrılmasını sağladığı için kelimelerin bir bütün olarak değil de kök ve ek bilgilerine ayrılarak kodlanması fikri ortaya atılmıştır. Bu üç yöntem *Heceye Dayalı Statik Huffman* (HeDSH), *Köke Dayalı Statik Huffman* (KökDSH) *ve Alınabilen İlk Büyük Heceye Dayalı Statik Huffman* (IHeDSH) algoritmaları olarak ele alınıp incelenecektir.

5.4.1 Heceye Dayalı Statik Huffman yöntemi

Heceye Dayalı Statik Huffman yöntemi de *Karaktere Dayalı Statik Huffman* yöntemine benzer. *Karaktere Dayalı Statik Huffman* yönteminde karakterler için oluşturulmuş statik şablon yerine, bu yöntemde hecelerden oluşan statik bir şablon oluşturulmuştur.

Kelimenin oluşumunda sesyolundan bir çırpıda çıkan sese *hece* denir. Heceler, birkaç ünlü-ünsüz sesin biraraya gelmesiyle oluşur. Ünlüler de tek başına bir kelimenin hecesi olabilirler.

Bir, iki, üç ve dört sesten oluşan heceler vardır.
Bir sesli hece, yalnız ünlülerden oluşur: a,e,ı,i,o,ö,u,ü
İki sesli hecelerde, bir ünlü bir ünsüz vardır. Her ikisi başta da sonda da olabilir: bu, iş...
Üç sesli hecelerde bir tane olan ünlü ya öndedir ya da aradadır: çok,ört...
Dört sesli hecelerde ise: «ünsüz-ünlü-ünsüz-ünsüz» koşulu vardır: kalk, türk...

5.4.1.1 Heceye dayalı statik şablonun oluşturulması

Heceler için statik şablonun (*hecesab*) oluşturulabilmesi için *test-kümesi* içerisinde yer alan bütün metin dosyalarındaki kelimeler hecelerine ayrılmış ve kullanılan bütün hecelerin sıklık bilgileri ile bir dosyada tutulmuşlardır. Hece şablonunu oluşturacak olan heceler, kullanım sıklığı en büyük olan hecenin yüzde birinden daha küçük sıklığa sahip olanlar hariç, geride kalanların hece şablonuna dahil edilmesiyle oluşturulur. Bu şablona, metin içerisinde geçen özel karakterlerin kullanım sıklıklarının da eklenmesiyle, 578 adet farklı sembolden oluşan bir şablon oluşturulur (EK 5). Şablon içerisinde semboller, Huffman kodları ve Huffman kodu bit uzunlukları bulunmaktadır.

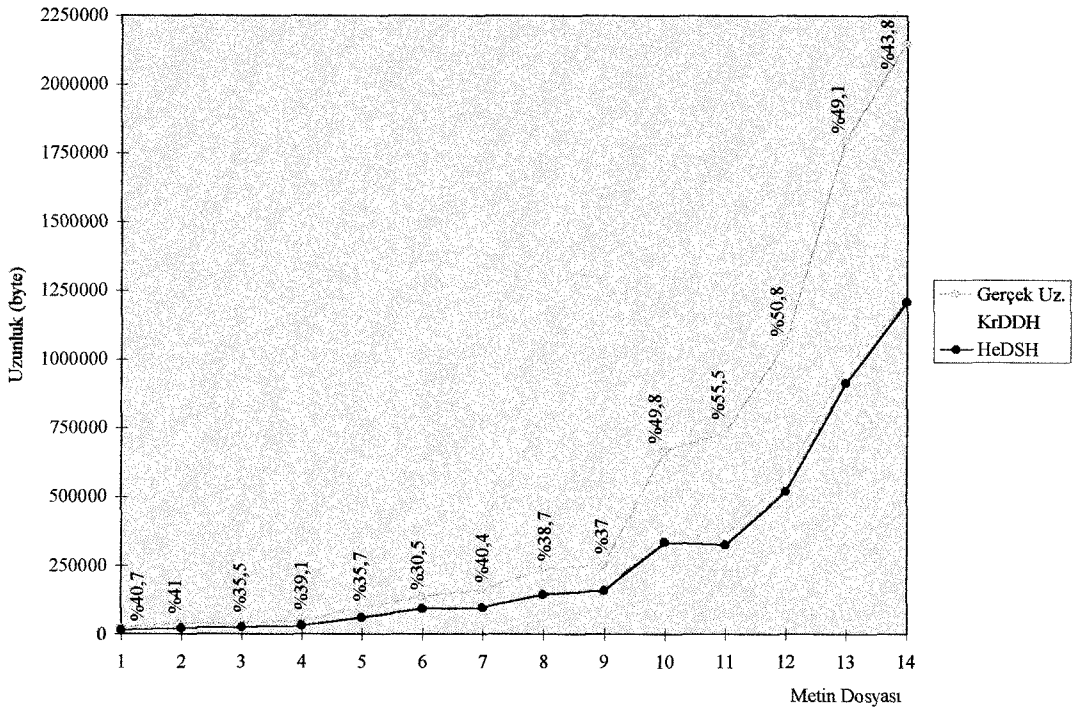
5.4.1.2 HeDSH yöntemi ile Türkçe bir metnin sıkıştırılması/genişletilmesi

Heceye Dayalı Statik Huffman yöntemi ile sıkıştırma işlemine başlamadan önce hece şablonunun belleğe çekilmesi gerekir. Bu işlem yaklaşık 220ms. sürmektedir. Mevcut hece şablonu kullanılarak sıkıştırılmak istenen bir metin dosyasında, her kelime öncelikli olarak hecelerine ayrılır. Aranan hece şablon içerisinde mevcut ise ilgili kod değerini alarak sıkıştırma işlemine devam edilir. Eğer şablon içerisinde bulunmayan bir hece ile karşılaşılırsa, hece öbeği harf harf kodlanır. *Heceye Dayalı Statik Huffman* yöntemi kullanılarak *test-kümesi* içerisindeki dosyalar sıkıştırıldığında, her dosyanın sıkıştırma verimi, sıkıştırma ve genişletme süreleri, gerçek metin dosyaları ile karşılaştırmalı olarak Çizelge 5.8'de verilmekte olup, Şekil 5.10'da da grafik olarak gösterilimi sunulmaktadır.

Çizelge 5.8'den *Heceye Dayalı Statik Huffman* yöntemi kullanılarak bir metin dosyasının sıkıştırılmasında, ortalama **%47** 'lik sıkıştırma verimi elde edilmiş olduğu gözlenmektedir. Bu yöntemde de genişletme süresi, sıkıştırma süresinin yaklaşık **2,7** katı daha fazla sürmektedir (GS = 2,7 * SS).

Çizelge 5.8 Gerçek metin dosyası ile HeDSH yönteminden alınan sonuçların karşılaştırılması

Sıra No	Dosya Adı	Gerçek Uzunluk	HeDSH	SS (ms)	η%	GS (ms)
1	Donat.txt	24.624	14.609	390	40,7	1.160
2	Gciva.txt	36.596	21.577	600	41,0	1.650
3	Takyol.txt	41.377	26.677	770	35,5	2.070
4	Masik.txt	48.881	29.747	880	39,1	2.300
5	Tubitak.txt	93.884	60.353	1.810	35,7	4.670
6	Phdthesis.txt	135.361	94.095	2.097	30,5	7.300
7	Ykemal.txt	160.240	95.537	2.910	40,4	7.470
8	Furuzan.txt	236.705	145.147	4.450	38,7	11.320
9	Omersey.txt	251.606	158.583	4.560	37,0	12.360
10	Oeksi.txt	666.120	334.319	9.880	49,8	26.260
11	Heper.txt	731.929	325.397	9.670	55,5	25.650
12	Ecolasan.txt	1.055.244	519.673	15.210	50,8	40.870
13	Mabirand.txt	1.795.226	913.397	27.620	49,1	71.730
14	Caltan.txt	2.147.095	1.207.723	34.870	43,8	94.690



Şekil 5.10 Gerçek metin dosyası ile KrDDH ve HeDSH yönteminin sıkıştırma verimlerinin karşılaştırılması

5.4.2 Köke Dayalı Statik Huffman yöntemi

Köke Dayalı Statik Huffman yöntemi de *Karaktere* ve *Heceye Dayalı Statik Huffman* yöntemlerine benzer. *Köke Dayalı Statik Huffman* yöntemi, karakterler ve heceler için oluşturulmuş statik şablonlar yerine, kelimenin kökünü ve eklerini oluşturacak olan kök ve eklerden meydana gelen bir şablona ihtiyaç duymaktadır.

5.4.2.1 Kök ve eklerden oluşan statik şablonun oluşturulması

Kök ve eklerden oluşan statik bir şablona sahip olabilmemiz için öncelikle herhangi bir kelimenin kökünü ve ekini ayırabilmemiz gerekmektedir.

Kelimelerin, ekler ayrıldıktan sonra geriye kalan ve bölünemeyen anlamlı bölümüne *kök* adı verilir. Türkçe'de kökler genellikle tek hecelidir: al-, gör-, tut- gibi. Bununla birlikte birkaç heceden oluşan kökler de vardır: ara-, ayak-, otuz- gibi.

İsim veya fiil türü kelimelerden yeni kelimeler türetmek veya kelimelerin cümle içindeki görevlerini belirtmek amacıyla kullanılan öğelere *ek* denir. Genel dilbilgisinde önek, içek ve sonek olmak üzere üç çeşit ek vardır. Türkçe bitişken bir dil olduğundan sadece sonek kullanılır.

Kelimenin kök ve ek analizi sırasında, referans alabileceğimiz kök ve eklerden oluşan sözlüklere ihtiyaç olacaktır. Türkçe'deki köklerin ve eklerin çıkarılması için sırasıyla (Eyüpoğlu Z., 1995), (Çotuksöken Y., 1991) ve (Hatiboğlu V., 1974) kaynaklarından yararlanılarak EK 6'daki Kök sözlüğü ve EK 7'deki Ek sözlüğü oluşturulmuştur. Diğer statik yöntemlerdeki şablon hazırlama işlemlerinde olduğu gibi *test-kümesi* içerisindeki tüm metin dosyalarındaki kelimeler, kök ve ek sözlüklerinin yardımıyla kök ve eklerine ayrılırlar. Kullanım sıklıkları ile birlikte ayrı ayrı dosyalarda tutulan bu kök ve ek istatistiksel bilgileri, kök ve ekteki ortak olarak kullanılan sembolleri belirlemek amacıyla yeniden düzenlenip, kullanım sıklığı en fazla olan sembolün yüzde birinin altında kalanlar hariç, özel işaretlerinde eklenmesiyle kök,ek ve özel işaretlerin toplamından oluşan, 478 elemanlı, statik bir şablon (*köksab*) elde edilir (EK 8).

5.4.2.2 KökDSH yöntemi ile Türkçe bir metnin sıkıştırılması/genişletilmesi

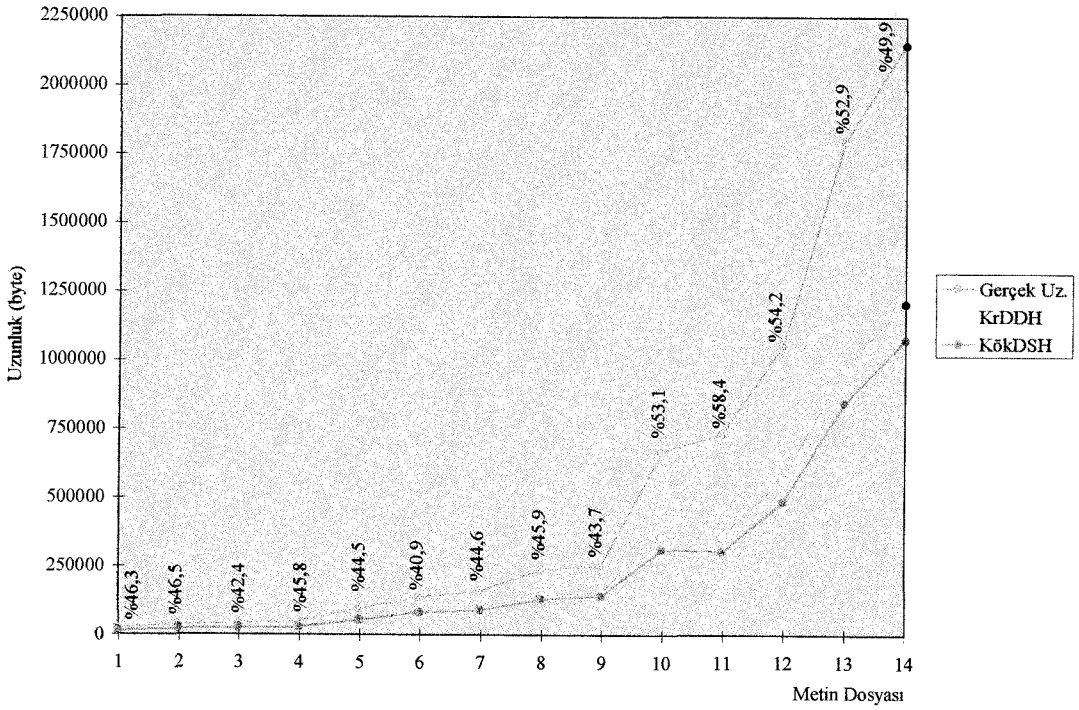
Köke Dayalı Statik Huffman yöntemi ile sıkıştırma işlemine başlamadan önce kök-ek şablonunun belleğe çekilmesi gerekir. Bu işlem yaklaşık 110ms. sürmektedir. Mevcut kök-ek şablonu kullanılarak sıkıştırılmak istenen bir metin dosyasında, her kelime öncelikli olarak kök ve eklerine ayrılır. Aranan kök veye ek, şablon içerisinde mevcut ise ilgili kod değerini alarak sıkıştırma işlemine devam edilir. Eğer şablon içerisinde yer almayan kök veya ek ile karşılaşılırsa, kök veya ek öbeği harf harf kodlanır. Çizelge 5.9'da *Köke Dayalı Statik Huffman* yöntemi kullanılarak sıkıştırılan, *test-kümesi* içerisindeki her dosyanın sıkıştırma

verimi, sıkıştırma ve genişletme süreleri, gerçek metin dosyaları ile karşılaştırmalı olarak verilmekte olup, Şekil 5.11'de de grafik olarak gösterilimi sunulmuştur.

Çizelge 5.9 Gerçek metin dosyası ile KökDSH yönteminden alınan sonuçların karşılaştırılması

Sıra No	Dosya Adı	Gerçek Uzunluk	KökDSH	SS (ms)	η%	GS (ms)
1	Donat.txt	24.624	13.225	330	46,3	820
2	Gciva.txt	36.596	19.577	550	46,5	1.270
3	Takyol.txt	41.377	23.815	600	42,4	1.540
4	Masik.txt	48.881	26.503	710	45,8	1.700
5	Tubitak.txt	93.884	52.075	1.540	44,5	3.350
6	Phdthesis.txt	135.361	79.947	2.014	40,9	5.016
7	Ykemal.txt	160.240	88.737	2.470	44,6	5.710
8	Furuzan.txt	236.705	128.071	3.680	45,9	8.300
9	Omersey.txt	251.606	141.667	2.900	43,7	9.120
10	Oeksi.txt	666.120	312.509	8.460	53,1	20.270
11	Heper.txt	731.929	304.133	8.240	58,4	19.800
12	Ecolasan.txt	1.055.244	483.337	12.960	54,2	31.120
13	Mabirand.txt	1.795.226	844.697	23.560	52,9	54.730
14	Caltan.txt	2.147.095	1.075.557	29.770	49,9	70.000

Çizelge 5.9'dan *Köke Dayalı Statik Huffman* yöntemi kullanılarak bir metin dosyasının sıkıştırılmasında, ortalama **%52** 'lik sıkıştırma verimi elde edilmiş olduğu gözlenmektedir. Bu yöntemde de genişletme süresi, sıkıştırma süresinin yaklaşık **2,4** katı daha fazladır (GS=2,4 * SS).



Şekil 5.11 Gerçek metin dosyası ile KrDDH ve KökDSH yönteminin sıkıştırma verimlerinin karşılaştırılması

5.4.3 İlk Heceye Dayalı Statik Huffman yöntemi

Sembollerin kullanılmasıyla gerçekleştirilen sıkıştırma yöntemlerinden üçüncüsü, *Alınabilen İlk Uzun Heceye Dayalı Statik Huffman* yöntemidir. Bu yöntemde kelimenin analizi yapılırken, kelimenin ilk hecesi yerine alınabiliniyorsa en uzun ilk hecesi alındıktan sonra, kelimenin arta kalan kısmıda eklerine ayrılarak analiz kısmı sonlandırılır. Örneğin: Analiz edilecek kelime **sınav** olsun. Bu kelimenin ilk hecesi **sı** dır. Eğer elimizde **sın** hecesine verilmiş bir kod var ise, kelimeyi **sı** yerine **sın**'dan itibaren kesip, analiz işlemine ekleri bularak devam ederiz.

5.4.3.1 Alınabilen ilk büyük hece ve eklerden oluşan statik şablonun oluşturulması

Diğer statik şablonların oluşturulmasında olduğu gibi, *İlk Heceye Dayalı Statik Huffman* yönteminin kullanacağı şablon için ek ve ilk hece sözlüklerine ihtiyaç duyulmaktadır. Bir önceki yöntemde ek sözlüğümüz oluşturulmuştu. En uzun ilk hecenin bulunabilmesi için de Türk Dil Kurumu imla kılavuzu ve sözlüğünden (Türk Dil Kurumu, 1996) yararlanılmıştır. Yapılan inceleme sonucunda 5032 adet farklı hece olabileceği fakat bu hecelerden 2019 tanesinin bizim kullandığımız *test-kümesi*'ndeki dosyaların, kullandıkları en uzun ilk hece olduğu saptanmıştır. Yapılan analiz sonucunda elde edilen hece ve ekler diğer şablonların oluşturulması sırasında izlenen adımlardan geçtikten sonra Ek 9'da verilen ilkhece-ek-sembol statik şablonu oluşturulmuştur.

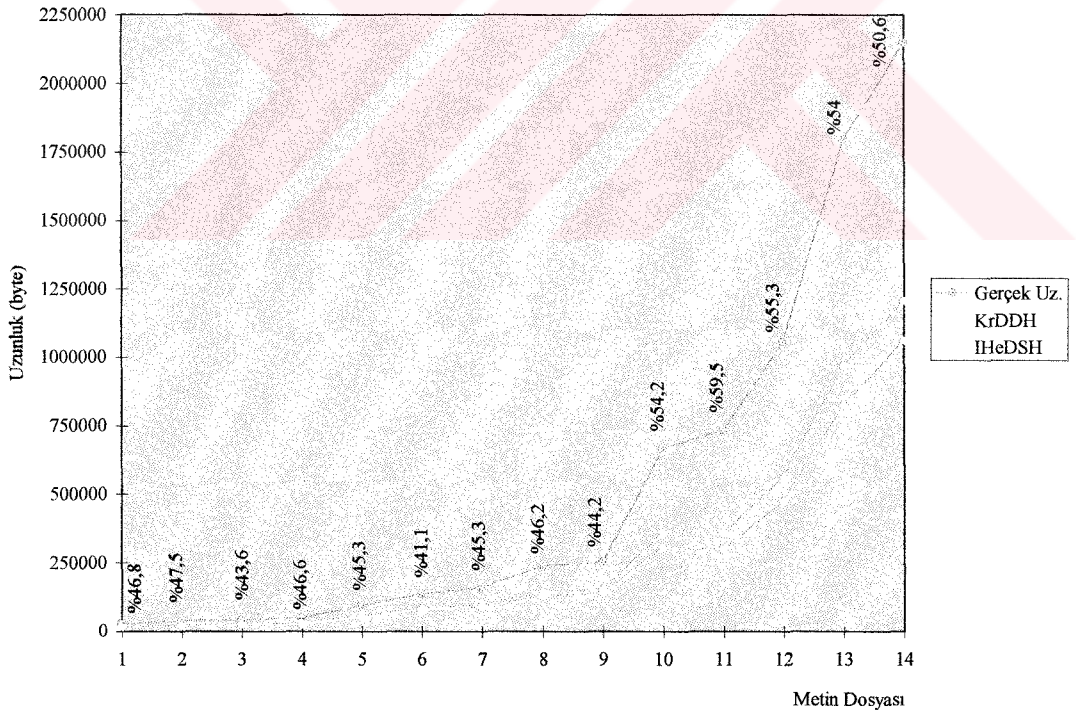
5.4.3.2 IHeDSH yöntemi ile Türkçe bir metnin sıkıştırılması/genişletilmesi

Alınabilen ilk uzun hece, ekler ve özel işaretlerden oluşturulmuş olan 533 elemanlı şablon kullanılarak (şablonun belleğe çekilmesi 110ms sürmektedir), *test-kümesi* içerisindeki dosyalar *İlk Heceye Dayalı Statik Huffman* yöntemi kullanılarak sıkıştırıldığında, dosyaların sıkıştırma verimi, sıkıştırma ve genişletme süreleri, gerçek metin dosyaları ile karşılaştırmalı olarak Çizelge 5.10'da verilmekte olup, Şekil 5.12'de de grafik olarak gösterilimi sunulmaktadır.

Çizelge 5.10'dan *İlk Heceye Dayalı Statik Huffman* yöntemi kullanılarak bir metin dosyasının sıkıştırılmasında, ortalama **%53** 'lük sıkıştırma verimi elde edilmiş olduğu gözlenmektedir. Bu yöntemde de genişletme süresi, sıkıştırma süresinin yaklaşık **2,5** katı daha fazladır (GS = 2,5 * SS).

Çizelge 5.10 Gerçek metin dosyası ile IHeDSH yönteminden alınan sonuçların karşılaştırılması

Sıra No	Dosya Adı	Gerçek Uzunluk	IHeDSH	SS (ms)	η%	GS (ms)
1	Donat.txt	24.624	13.109	280	46,8	880
2	Gciva.txt	36.596	19.231	490	47,5	860
3	Takyol.txt	41.377	23.351	600	43,6	1.540
4	Masik.txt	48.881	26.085	710	46,6	1.750
5	Tubitak.txt	93.884	51.337	1.430	45,3	3.400
6	Phdthesis.txt	135.361	79.763	2.150	41,1	5.027
7	Ykemal.txt	160.240	87.647	2.360	45,3	5.880
8	Furuzan.txt	236.705	127.387	3.630	46,2	8.570
9	Omersey.txt	251.606	140.281	3.730	44,2	9.400
10	Oeksi.txt	666.120	304.761	8.070	54,2	20.550
11	Heper.txt	731.929	296.385	7.070	59,5	20.010
12	Ecolasan.txt	1.055.244	471.483	12.360	55,3	31.750
13	Mabirand.txt	1.795.226	825.041	22.460	54,0	55.690
14	Caltan.txt	2.147.095	1.061.379	28.450	50,6	71.510



Şekil 5.12 Gerçek metin dosyası ile KrDDH ve IHeDSH yönteminin sıkıştırma verimlerinin karşılaştırılması

5.5 KrDDH, HeDSH, KökDSH ve IHeDSH Yöntemlerinin Karşılaştırılması

Bu bölümde *Karaktere Dayalı Dinamik Huffman* yöntemi ile *Heceye Dayalı Statik Huffman*, *Köke Dayalı Statik Huffman* ve *İlk Heceye Dayalı Statik Huffman* yöntemleri karşılaştırmalı olarak antatılmaktadır. Bu dört yöntemin sıkıştırma verimleri Çizelge 5.11'de, grafik olarak gösterilimide Şekil 5.13'de sunulmuştur.

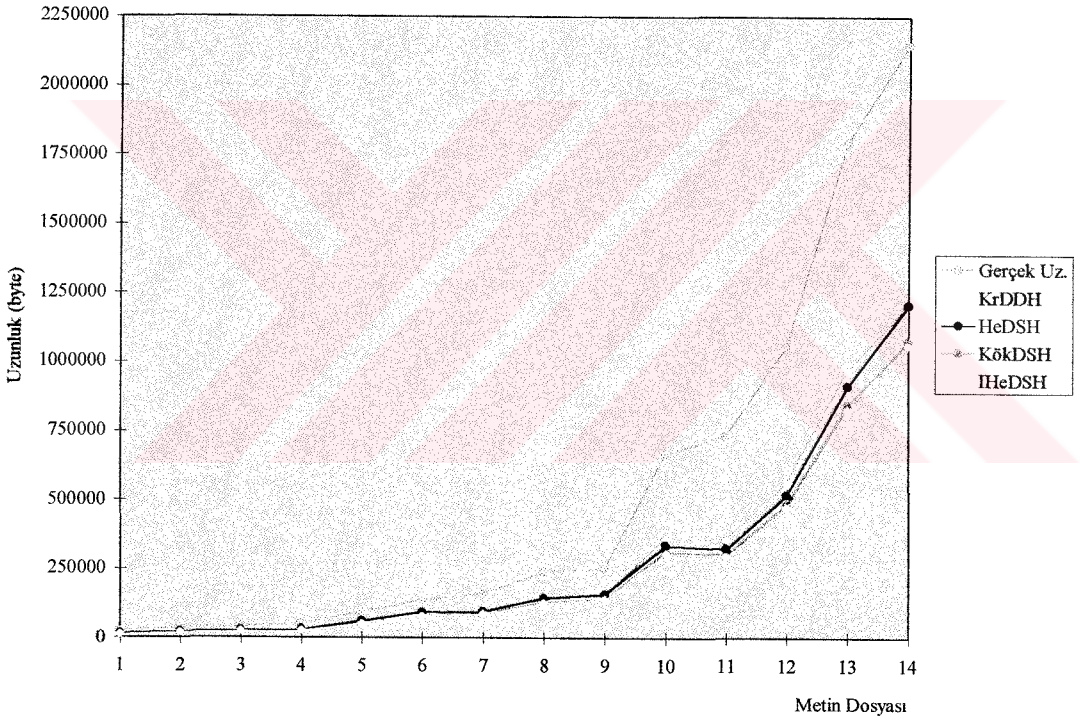
Karaktere Dayalı Dinamik Huffman yöntemi test-kümesi içerisindeki dosyaları ortalama **%45**'lik bir başarı ile sıkıştırmaktadır. Karakter bazında işlem gördüğünden, sıkıştırma ve genişletme süreleri diğer yöntemlere göre daha az sürmektedir. *Karaktere Dayalı Dinamik Huffman* yöntemi, büyük metinlerden olan "caltan.txt" dosyasını *İlk Heceye Dayalı Statik Huffman* yöntemine göre ortalama **3** katı daha hızlı olarak sıkıştırmaktadır. Genişletme süreleri ise birbirine daha yakındır. Buna karşılık, sıkıştırma verimi *İlk Heceye Dayalı Statik Huffman* yönteminde ortalama **%8** daha fazladır.

Çizelge 5.11 Gerçek metin, KrDDH, HeDSH, KökDSH ve IHeDSH yöntemlerinden alınan başarım oranları

	KrDDH			HeDSH			KökDSH			IHeDSH		
Sıra No	η %	SS (ms)	SS/GS (ms)	η %	SS (ms)	SS/GS (ms)	η %	SS (ms)	SS/GS (ms)	η %	SS (ms)	SS/GS (ms)
1	**40,0**	170	0,26	**40,7**	390	0,34	**46,3**	330	0,40	**46,8**	280	0,32
2	**40,4**	220	0,21	**41,0**	600	0,36	**46,5**	550	0,43	**47,5**	490	0,57
3	**38,1**	220	0,18	**35,5**	770	0,37	**42,4**	600	0,39	**43,6**	600	0,39
4	**40,1**	280	0,20	**39,1**	880	0,38	**45,8**	710	0,42	**46,6**	710	0,41
5	**39,9**	490	0,18	**35,7**	1.810	0,39	**44,5**	1.540	0,46	**45,3**	1.430	0,42
6	**39,3**	720	0,18	**30,5**	2.097	0,29	**40,9**	2.014	0,40	**41,1**	2.150	0,43
7	**39,9**	880	0,20	**40,4**	2.910	0,39	**44,6**	2.470	0,43	**45,3**	2.360	0,40
8	**40,6**	1.260	0,19	**38,7**	4.450	0,39	**45,9**	3.680	0,44	**46,2**	3.630	0,42
9	**39,6**	1.370	0,19	**37,0**	4.560	0,37	**43,7**	2.900	0,32	**44,2**	3.730	0,40
10	**46,2**	3.300	0,18	**49,8**	9.880	0,38	**53,1**	8.460	0,42	**54,2**	8.070	0,39
11	**51,4**	3.360	0,19	**55,5**	9.670	0,38	**58,4**	8.240	0,42	**59,5**	7.070	0,35
12	**46,9**	5.280	0,20	**50,8**	15.210	0,37	**54,2**	12.960	0,42	**55,3**	12.360	0,39
13	**46,1**	490	0,18	**49,1**	27.620	0,39	**52,9**	23.560	0,43	**54,0**	22.460	0,40
14	**43,9**	11.580	0,20	**43,8**	34.870	0,37	**49,9**	29.770	0,43	**50,6**	28.450	0,40

Sembol sıkıştırma yöntemlerinde, sıkıştırma verimi *Heceye Dayalı Statik Huffman*, *Köke Dayalı Statik Huffman* ve *İlk Heceye Dayalı Statik Huffman* sırasında artış göstermektedir. *Heceye Dayalı Statik Huffman* yöntemi sıkıştırma veriminde ortalama **%47**, *Köke Dayalı Statik Huffman* yöntemi sıkıştırma veriminde ortalama **%52**, *İlk Heceye Dayalı Statik Huffman* yöntemi de ortalama **%53**'e varan bir başarı göstermiştir. Bu sebepden dolayı ileri bölümlerde anlatılacak olan yöntemler *İlk Heceye Dayalı Statik Huffman* yöntemi ile karşılaştırmalı olarak verilecektir.

Sıkıştırma sürelerinde meydana gelen farklılıklar, kelimenin nasıl analiz edildiğine ve aranan hece,ek veya kökün kaçıncı erişimde bulunduğuna bağlı olarak değişmektedir.



Şekil 5.13 Gerçek metin, KrDDH, HeDSH, KökDSH ve IHeDSH yöntemlerinin karşılaştırılması

Heceye Dayalı Statik Huffman yöntemi, *Karaktere Dayalı Dinamik Huffman* yöntemine göre sıkıştırma veriminde **%2** daha iyi sonuç vermiştir. Hecelerin ortalama uzunluğunun üç karakter olduğunu kabul edersek, daha iyi bir sonuç almamız beklenirdi. Fakat Huffman kodlama ağacının eleman sayısı, *Karaktere Dayalı Dinamik Huffman* yöntemindeki

kodlama ağacının eleman sayısından daha fazla olduğundan ağacın seviyesi artmaktadır. Bu da hecelerin daha fazla bit ile kodlanmasına neden olmaktadır. *Köke Dayalı Statik Huffman* yöntemi, *Heceye Dayalı Statik Huffman* yöntemine göre sıkıştırma veriminde ortalama **%5**'lik daha iyi bir başarı vermiştir. Bunun sebebi, her iki yöntemin kullanmış oldukları şablonlardaki eleman sayıları arasındaki farklılıkdır. Ayrıca, hecelerin kullanım sıklıklarından daha fazla sıklıkla kullanılan eklerin mevcut olmasıdır. Sıklık değeri ne kadar fazla olursa, Huffman kodunun bit uzunluğu ters orantılı olarak değişmektedir.

İlk Heceye Dayalı Statik Huffman yöntemi, *Köke Dayalı Statik Huffman* yöntemine göre sıkıştırma veriminde **%1**'lik daha fazla başarı sağlamıştır. Daha iyi sonuç alınmasının sebebi, kelime köklerinin bir kısmının iki karakter ile ifade edilmesine karşılık, *İlk Heceye Dayalı Statik Huffman* yönteminde alınabilen ilk en uzun hecenin, büyük bir çoğunlukla üç karakterden oluşmasıdır.

5.6 Türkçe Metinde Kelime Sıkıştırma

Türkçe'nin biçimbilimsel yapısını temel alarak geliştirmeye çalıştığımız sıkıştırma yönteminin başarımını daha da arttırabilmek amacıyla, daha çok bilgiyi bir seferde kodlayabilmek için, kelimeleri sadece kök ve eklerine ayırmak yerine kök-gövde ve eklerine ayırma işleminin yapılması uygun görülmüştür. İsim veya fiil köklerine birtakım yapım ekleri getirilerek oluşturulan türeve *gövde* adı verilir (aç-lık, bak-ı-m, gör-ü-ş gibi). Kelimelerin kök yerine, gövde ve eklerine ayrılarak analiz edilmesiyle daha fazla bilginin bir seferde kodlanması mümkün olmaktadır Kelimelerin biçimbilimsel olarak incelenmesinde, bir kökün bir gövde tarafından kapsanması durumunda kelime, kökü yerine gövdesinden ayrılmaktadır. Örneğin; "*ay*" ve "*aydın*" kelimelerinin her ikisi de sözlükte verilmiş olsun. Ayırma işlemine girecek olan kelime de "*aydınlık*" ise, kök olarak "*ay*" yerine "*aydın*" kelimesi seçilmelidir. Yani "aydınlık" kelimesi; **aydın:gövde**, **lık:ek** olarak değerlendirilmelidir. Bu yöntem yukarıda anlatılan, statik şablon kullanan, yöntemlere göre kök-gövde ve eklerin Türkçe'deki genel kullanım sıklıkları hakkında önceden hazırlanmış bir bilgiye ihtiyaç duymaz. Kök-gövde ve eklerin kullanım sıklıkları, sıkıştırılmak üzere işlenmekte olan metin dosyası içerisinden, sadece o dosyaya bağlı kalınarak elde edilir. Böylece aynı kelimeye, geçtiği her metin

dosyasının karakteristiğine en uygun Huffman kod kelimesinin atanması sağlanır. Bu açıdan oluşturulan yöntem dinamik Huffman algoritması olarak değerlendirilebilir.

5.6.1 Kök-gövde ve ek sözlüklerinin hazırlanması

Kelimenin kök-gövde ve eklerine ayrılabilmesi için kök-gövde ve ekler için iki ayrı sözlüğe ihtiyacımız olacaktır. Kök-gövde sözlüğünün oluşturulmasında Türkçe sözlük (Türk Dil Kurumu, 1992) ve imla kılavuzundan (Türk Dil Kurumu, 1996) yararlanılarak **13.206** adet kök ve gövde içeren kök sözlüğü oluşturulmuştur. Ek sözlüğünün oluşturulmasında da (Eyüpoğlu Z., 1995), (Çotüksöken Y., 1991) ve (Hatiboğlu V., 1974) kaynaklarından yararlanılarak yapım ve çekim eklerinin (522 adet) hepsini içine alan, bunlara ek olarak özel işaret ve kelime içerisinde geçen büyük-küçük harf ayırımı için kullanılacak olan kontrol bilgileriyle birlikte **580** elemanlı bir ek sözlüğü oluşturulmuştur. Kök ve ek sözlüklerinde yer alan bilgilerin hepsi küçük harf olarak tutulmakta, metnin analizi de küçük harfe göre yapılmakta ve kodlanmaktadır. Sıkıştırılmış bir dosyanın geri açılması durumunda, orijinalliğini kaybetmemesi için, kelime içerisindeki hangi harf veya harflerin büyük olduğu bilgisinin tutulması gerekir. Bunun için ek sözlüğünün içinde iki farklı kontrol karakteri bulunmaktadır. Bu kontrol karakterleri, kelimenin tamamının büyük harf veya içerisinde bir yada daha fazla sayıda büyük harf bulunması halinde, kelime içerisindeki yerlerinin belirlenmesinde kullanılır.

5.6.2 Kelimeye Dayalı Dinamik Huffman yöntemi

Kelimeye Dayalı Dinamik Huffman yönteminde, kelime analizi sırasında elde edilen kök-gövde ve ek bilgileri "**G**" (kök-gövde) ve "**E**" (ek) ismini verdiğimiz iki farklı kayıt yapısında tutulmaktadır (Şekil 5.14). Yapılan araştırmada Türk dilinde kök ve gövdelerin 14 karakteri, eklerin ise 7 karakteri geçmediği belirlenmiş ve adı geçen "G" ve "E" ağaçları Huffman kodlarını oluşturma, metni sıkıştırma ve sıkıştırılmış metni açmada kullanılmaktadır.

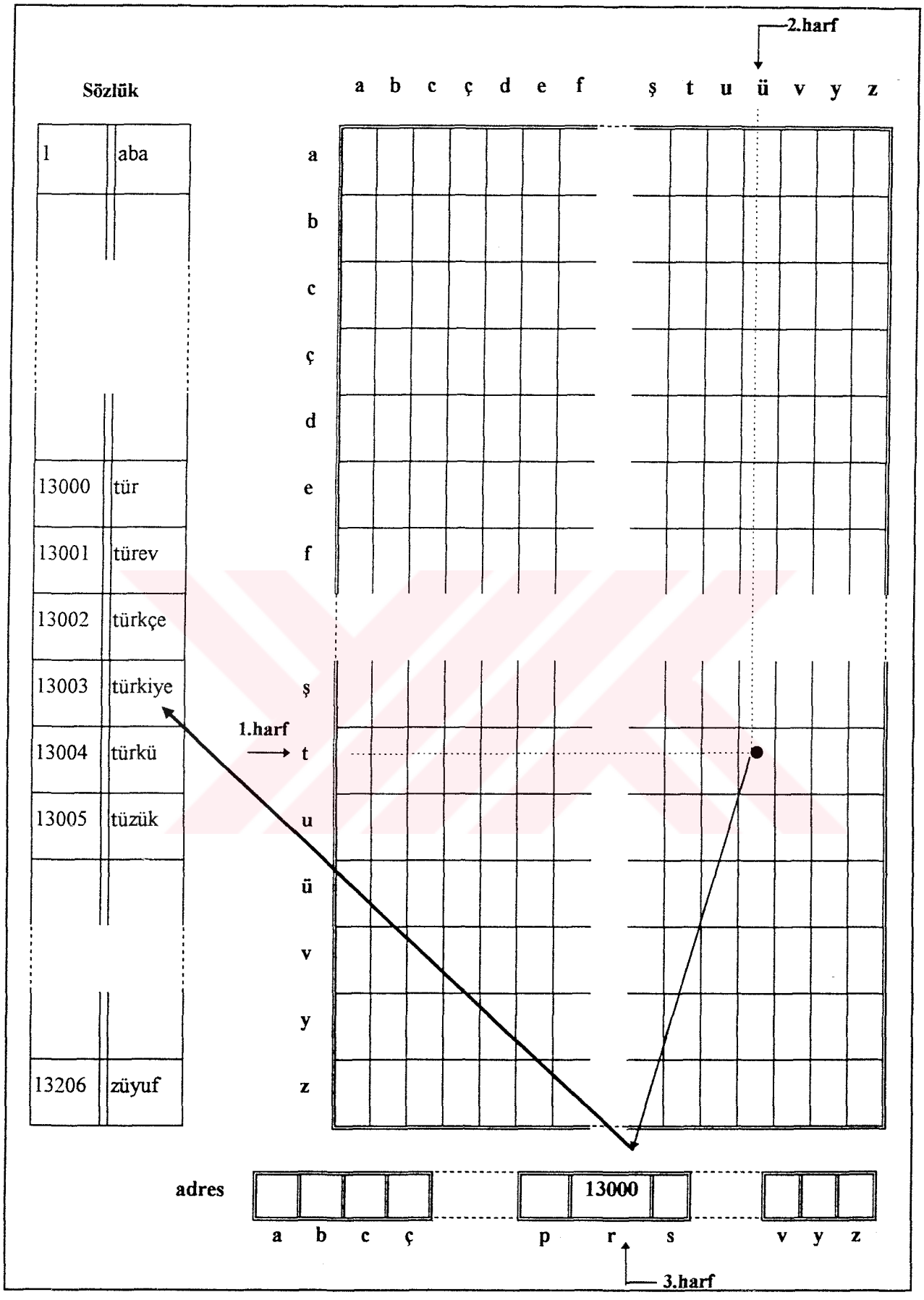
G kayıt yapısı	**E kayıt yapısı**
struct = RECORD	struct = RECORD
name : str14;	name : str7;
address : word;	address : word;
count : longint;	count : longint;
hcode : longint;	hcode : longint;
hlen : byte;	hlen : byte;
end;	end;

Şekil 5.14 Gövde ve ek bilgilerinin tutulduğu kayıt yapısı

Bu kayıt yapısında:

name	:	kök-gövde veya ekin ismi.
address:		kök-gövde veya ekin sözlükteki sırası.
count	:	kök-gövde veya ekin metin içerisindeki kullanım sıklığı.
hcode	:	kök-gövde veya ekin Huffman kodunun decimal değeri.
hlen	:	kök-gövde veya ekin Huffman kodunun bit uzunluğu.

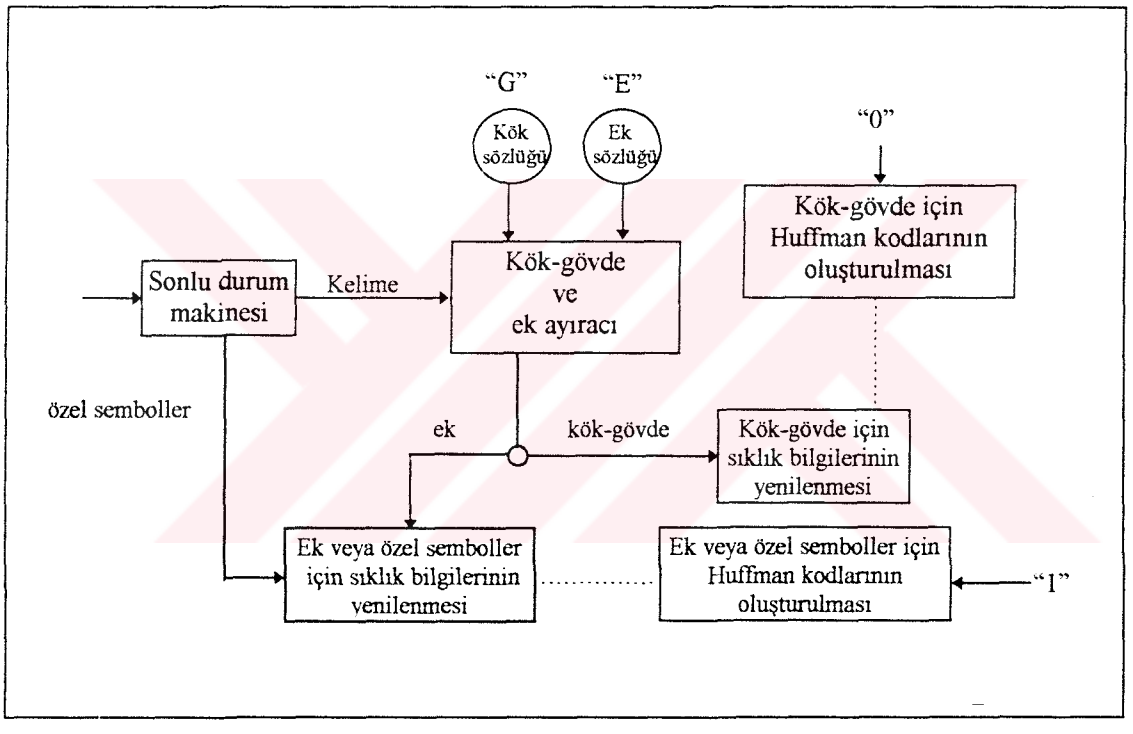
Kelimeye Dayalı Dinamik Huffman yönteminde kök ve ek sözlüklerinde aramayı kolaylaştırmak ve hızlandırmak amacıyla bir arama-tablosu'ndan (*look-up table*) yararlanılmıştır. Geliştirdiğimiz sıkıştırma algoritmasını içeren program çalıştırıldığında, kök-gövde ve ek sözlükleri belleğe çekilmektedir. Bu sözlükler alfabetik sıraya göre küçükten büyüğe doğru sıralı olduklarından, belleğe okunurken aynı zamanda da arama işlemlerini hızlandırmak için, kök-gövde'lerin iki karakterden oluşanlarının sözlükteki başlangıç adresleri iki boyutlu, üç veya daha fazla karakterden oluşan kök-gövde'lerin sözlükteki başlangıç adresleri de üç boyutlu bir diziye yerleştirilirler. Aynı işlem eklerin bir, iki, üç ve daha fazla karakterden oluşanlarının da sözlükteki başlangıç adresleri sırasıyla bir, iki ve üç boyutlu dizilere yerleştirilir (Şekil 5.15). Örneğin; ***türkiye*** kelimesi kök-gövde sözlüğünde arandığında, üç boyutlu diziden "***tür***" ile başlayan kelimelerin sözlükteki başlangıç adresi alınır. Bu örnek için sözlükteki başlangıç adres değeri **13000**' dir. Arama işlemi kelimenin üçüncü karakterinin değişimine kadar devam eder. Bu örnekte *türkiye* kelimesinin adresi **13003** dür.



Şekil 5.15 Arama tablosu

Kök-gövde ve ek sözlüklerinin belleğe çekilmesinden sonra sıkıştırılacak metin, bloklar halinde belleğe alınmakta ve her blok karakter karakter incelenmektedir. Bir bloğun bitiminde, otomatik olarak bir sonraki blok belleğe okunmakta ve bir kelimenin bir blokta başlayıp diğer blokta bitmesi durumunu hatasız olarak değerlendirebilmektedir.

Blokların karakter karakter değerlendirilmesini hızlandırmak ve uygulamayı modüler yapmak amacıyla bir sonlu durum makinesi geliştirilmiştir. Kullanılan bu makinenin çıkışından Huffman ağaçlarının girişine kadar geçen aşamalar Şekil 5.16'daki blok diyagramında gösterildiği gibi gerçekleştirilmiştir.



Şekil 5.16 Metnin analizinde ve kod kelimelerin oluşturulmasında izlenen yolun blok diyagramı

Sonlu durum makinesi, blok içerisinde bir kelimeyi yakaladığında, gerekli algoritmaları çağırarak, bu kelimenin en uygun şekilde kök-gövde ve eklerine ayrılmasını sağlamaktadır. Aynı zamanda kelimeler, küçük harfe çevrilerek analiz yapıldığından, kelime içerisinde geçen büyük harflerin geri dönüş işlemi sırasında hatasız olarak değerlendirilebilmesi için gerekli olan kontrol bilgilerinin de oluşturulması sağlanır. Elde edilen her kök-gövde bilgisi "G", ek, özel işaretler ve kontrol karakterleri "E" adını verdiğimiz kayıt yapılarına gönderilir.

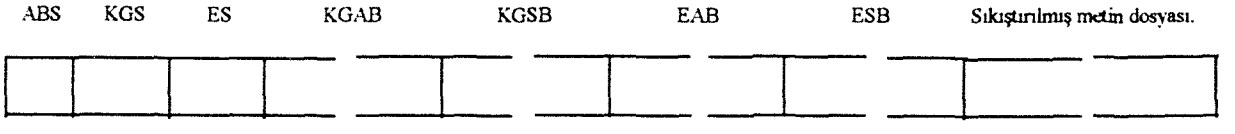
Eğer gönderilen bilgi daha önceden ilgili kayıt yapısına girilmiş ise sadece kullanım sıklık bilgisi arttırılır. Eğer gelen yeni bir bilgi ise kendisi ve sözlük adresi, ilgili kayıt yapısına eklenerek işleme dosya sonuna kadar devam edilir.

Ayrıca analiz sırasında kullanılan kök-gövde sözlüğünde yer almayan Türkçe veya yabancı bir kelime ile karşılaşılacak olunursa bu kelime karakter karakter kodlanır. Bu sayede oluşturulan sistem sıkıştırma işlemi sırasında, karşılaştığı özel durumlara da en iyi şekilde uyum sağlamaya çalışmaktadır.

Oluşan "G" ve "E" kayıt yapılarında bulunan sıklık bilgilerinden yararlanılarak iki Huffman kodlama ağacı oluşturulur. Biri kök-gövde diğeri de ekler için Huffman kodlarını üretir. Her iki ağaçda Huffman kodu açısından çakışmaların olması doğal olduğu için kod çözme işlemi sırasında karışıklığı önlemek amacıyla, "G" ağacında oluşan Huffman kodlarının başına "0" , "E" ağacında oluşan Huffman kodlarının başına da "1" biti eklenmiştir.

Huffman kodlarıyla kodlanmış bir bilgiyi geri açarken de yine "G" ve "E" ağaçlarından yararlanılır. Kodlanmış bit katarının ilk biti, kodun hangi Huffman ağacından çözüleceğini gösterir. "0" ise "G" ağacı, "1" ise "E" ağacı kullanılacaktır. Bu bit kontrolünden sonra ilgili ağacın en kısa kod uzunluğu referans alınarak bit katarı üzerinden aramaya başlanır. Eğer bu kod değerine sahip kök-gövde veya ek ile eşleşme sağlanırsa arkasından gelen bit yine "G" veya "E" ağacından hangisinin işleme gireceğine karar vermek için kullanılır. Eğer eşleşme sağlanamaz ise aranacak olan kodun uzunluğu bir arttırılarak işleme bir eşleşme sağlanıncaya kadar devam edilir.

Kodlanarak sıkıştırılmış bilginin kod çözücü tarafından kayıpsız olarak açılabilmesi için, sıkıştırılmış bilginin üzerine eklenen başlık (*header*) bilgisine ihtiyaç vardır. Bu çalışmada kullanılan başlık bilgisinin formatı Şekil 5.17'de verilmiştir.



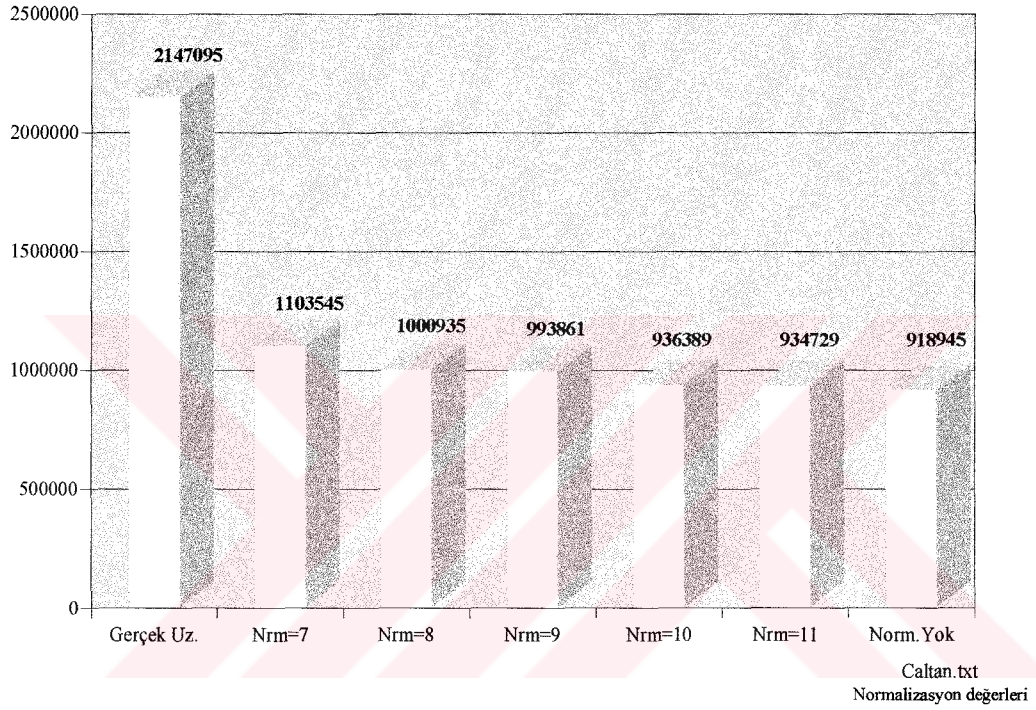
Şekil 5.17 Başlık formatı

ABS, Artık-bit-sayısı, bir byte uzunluğunda olup, kodlama sonucundaki son değerin sekiz bitten daha küçük olduğu durumlarda, kodu çözülecek son byte içerisindeki değerin kaç bitinin gerçek bilgi olmadığını göstermektedir. Metnin analizi sonucunda bulunan kök-gövde sayısı KGS alanında, ek sayısı da ES alanında tutulur. Kök-gövde adres bilgilerinin bulunduğu alan, KGAB, metin dosyasında geçen ve sözlükte yer alan her kök-gövde'nin sözlük içerisindeki adresini belirtir. Kullanılan kök-gövde sözlüğünde 13.206 adet kelime olduğu için her birinin adres bilgisi $\lceil \log_2 13.206 \rceil$ bit ile ifade edilmektedir. Başlık içerisinde bu alanın toplam uzunluğu $\lceil 14 * (\text{metin dosyasında geçen kök-gövde sayısı}) / 8 \rceil$ byte olacaktır. Ek adres bilgisi alanı, EAB ise eklerin adres bilgilerini tutar. Ek sözlüğünde toplam 580 adet ek bilgimiz olduğundan, her bir ek $\lceil \log_2 580 \rceil$ bit ile ifade edilebilir ve ek adres bilgisi alanı $\lceil 10 * (\text{metin dosyasında geçen ek sayısı}) / 8 \rceil$ byte uzunluğundadır. Kök-gövde'ye ait sıklık bilgisi KGSB alanında, eklere ait sıklık bilgisi de ESB alanında tutulmaktadır. Bu alanların byte olarak uzunlukları, sıklıkları ifade etmek için yapılan normalizasyona göre değişmektedir. Bu normalizasyon işlemi, gönderilecek olan sıklık bilgisinin daha az bit ile ifade edilebilmesi amacıyla yapılmıştır. Sıklık değerleri sırası ile (0-127), (0-254), (0-511), (0-1023) ve (0-2047) arasına denk düşecek şekilde normalize edilir. Normalizasyon işlemi, sıklık değerlerinin normalizasyon katsayısı ile çarpımı şeklinde gerçekleşir. Bu işlem bir aralık daraltma işlemi olduğu için, normalisazyon katsayısı $k < 1$ dir. En büyük sıklık değeri SÜ ise normalizasyon katsayısı,

$$k^{-1} = SÜ / 2^{nrm} \qquad nrm = 7,8,...,11$$

olarak hesaplanır. Normalizasyon işlemi metin dosyasının uzunluğuna göre farklı sonuçlar vermektedir. Küçük metinlerde, normalizasyon aralığının daraltılması veya genişletilmesi durumunda sıkıştırma veriminde **%1,5**'luk bir artış izlenmiştir. Metinlerin uzunluğu kısa olduğundan içerisinde geçen kök-gövde ve eklerin kullanım sıklıkları sayısı fazla değildir. Bu sebepten, normalizasyon aralığının geniş tutulması, sonucu fazla etkilememektedir. Büyük metinlerde ise durum tam tersidir. Normalizasyon aralığının daraltılması ve genişletilmesi durumunda sıkıştırma veriminde ortalama **%9**'luk bir artış izlenmiştir. Sıkıştırma veriminde meydana gelen bu artışın sebebi, büyük metin dosyalarında kök-gövde ve eklerin kullanım sıklık değerlerinin çok büyük olması ve normalizasyon aralığının geniş

tutulması durumunda daha düzgün bir dağılımın elde edilmesidir. Normalizasyon aralığının daraltılmasında sıkıştırma veriminde ki başarı düşmektedir. Çünkü, kullanım sıklıkları arasındaki farkın büyük olmasına rağmen, aralık içerisinde aynı noktaya düşen kök-gövde sayısının fazla olması sebebi ile, kodlama yeterince efektif olamamaktadır. Şekil 5.18, tek bir metin dosyası için farklı normalizasyon aralıklarında ve hiç normalizasyon yapılmadan gerçekleştirilen sıkıştırma işleminden sonraki dosya boylarını göstermektedir.



Şekil 5.18 Normalizasyon aralığı (0-127),(0-254),(0-511),(0-1023),(0-2047) arasında tutultuğunda alınan sıkıştırma değerleri

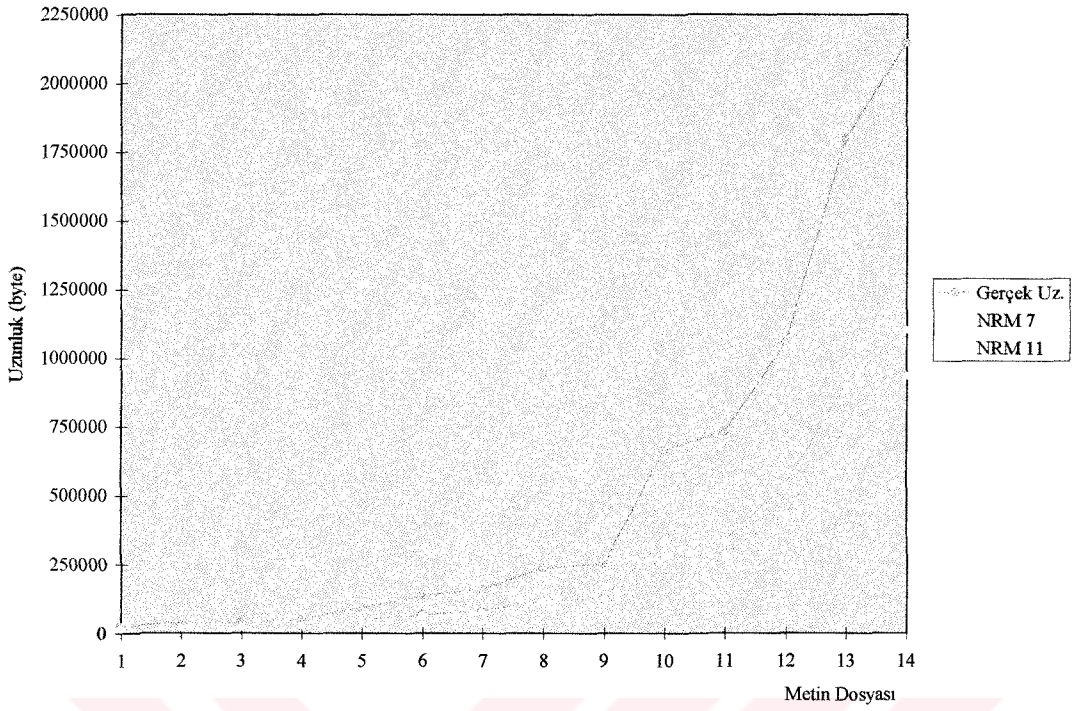
Normalizasyon sınır aralığını (0-2047) arasında aldığımızda, kök-gövde ve ek sıklık bilgilerinin kapladığı alan ⌈(kök-gövde sayısı + ek sayısı) * 11 bit / 8⌉ byte uzunluğundadır. *Kelimeye Dayalı Dinamik Huffman* algoritması ile bir metin dosyasını sıkıştırırken, metin içerisinde geçen kök-gövde ve ek'lerin kullanım sıklık bilgileri, normalizasyon aralığının farklı alınması durumunda ayrı ayrı kodlanmış olup, *test-kümesi* içerisindeki bütün dosyaların *nrm* değişkeninin 7 ve 11 alınması halinde elde edilmiş olan sıkıştırma verimleri karşılaştırmalı olarak Çizelge 5.12'de verilmiş ve Şekil 5.19'da da grafik olarak

gösterilmiştir. *Nrm* değişkeninin 7 alınması ile 11 alınması arasında, sıkıştırma veriminde ortalama **%7,5**'luk bir artış izlenmektedir.

Çizelge 5.12 Normalize değer aralığının değişmesiyle sıkıştırma verimleri arasındaki fark

Sıra No	Dosya Adı	Gerçek Uzunluk	NRM 7	η%	NRM 11	η%
1	Donat.txt	24.624	13.791	44,0	13.637	44,6
2	Gciva.txt	36.596	19.831	45,8	19.397	47,0
3	Takyol.txt	41.377	23.029	44,3	22.629	45,3
4	Masik.txt	48.881	25.503	47,8	25.155	48,5
5	Tubitak.txt	93.884	48.625	48,2	47.597	49,3
6	Phdthesis.txt	135.361	65.209	51,8	62.663	53,7
7	Ykemal.txt	160.240	82.071	48,8	77.851	51,4
8	Furuzan.txt	236.705	118.169	50,1	114.481	51,6
9	Omersey.txt	251.606	128.741	48,8	122.965	51,1
10	Oeksi.txt	666.120	338.659	49,2	279.421	58,1
11	Heper.txt	731.929	334.485	54,3	284.371	61,1
12	Mabirand.txt	1.795.226	899.871	49,9	738.093	58,9
13	Ecolasan.txt	1.055.244	531.987	49,6	430.409	59,2
14	Caltan.txt	2.147.095	1.103.545	48,6	934.729	56,5

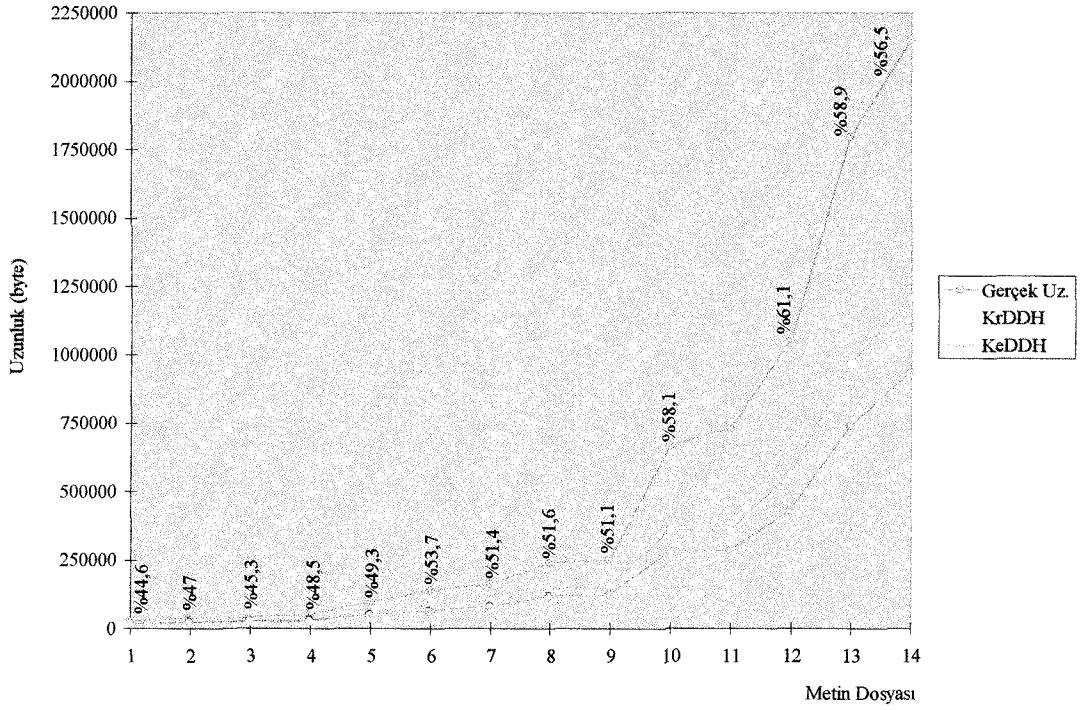
Kök-gövde ve ek bilgilerinin kullanım sıklıklarının, normalizasyon aralığının (0-2047) arasında alınması ile *Kelimeye Dayalı Dinamik Huffman* yönteminden elde edilen sonuç değerlerinin sıkıştırmadan önceki dosya boyları, sıkıştırma verimleri, sıkıştırma ve genişletme süreleri karşılaştırılmalı olarak Çizelge 5.13'de verilmekte olup, Şekil 5.20'de de grafik olarak gösterilmiştir.



Şekil 5.19 Normalizasyon aralıklarının farklı alınmasıyla elde edilen sıkıştırma başarımının karşılaştırılması

Çizelge 5.13 Gerçek metin dosyası ile KeDDH yönteminden alınan sonuçların karşılaştırılması

Sıra No	Dosya Adı	Gerçek Uzunluk	KeDDH	SS (ms)	η%	GS (ms)
1	Donat.txt	24.624	13.637	2.030	44,6	2.410
2	Gciva.txt	36.596	19.397	2.800	47,0	3.400
3	Takyol.txt	41.377	22.629	3.790	45,3	4.450
4	Masik.txt	48.881	25.155	4.170	48,5	5.000
5	Tubitak.txt	93.884	47.597	9.060	49,3	10.490
6	Phdthesis.txt	135.361	62.663	4.780	53,7	6.470
7	Ykemal.txt	160.240	77.851	10.270	51,4	12.530
8	Furuzan.txt	236.705	114.481	22.850	51,6	26.690
9	Omersey.txt	251.606	122.965	25.100	51,1	29.220
10	Oeksi.txt	666.120	279.421	39.270	58,1	46.950
11	Heper.txt	731.929	284.371	28.460	61,1	36.740
12	Ecolasan.txt	1.055.244	430.409	58.270	59,2	69.480
13	Mabirand.txt	1.795.226	738.093	77.880	58,9	103.700
14	Caltan.txt	2.147.095	934.729	131.230	56,5	165.930



Şekil 5.20 Gerçek metin dosyası ile KrDDH ve KeDDH yönteminin sıkıştırma verimlerinin karşılaştırılması

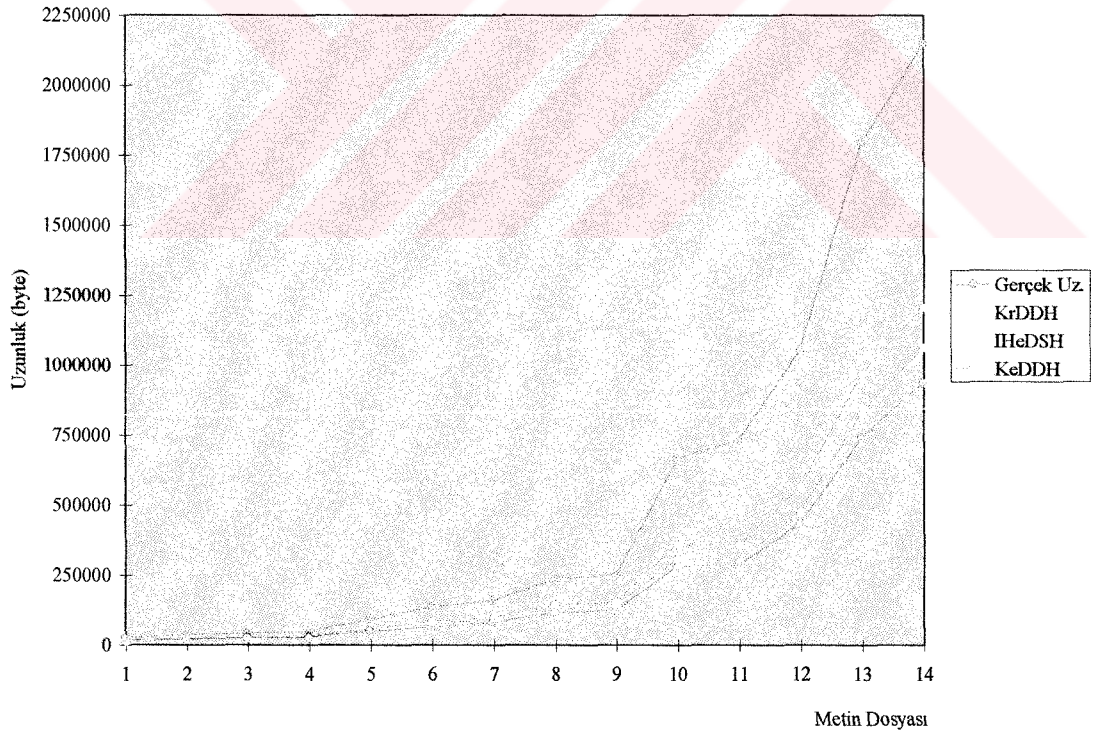
Çizelge 5.13'den *Kelimeye Dayalı Dinamik Huffman* yöntemi kullanılarak bir metin dosyasının sıkıştırılmasında, ortalama **%57** 'lik sıkıştırma verimi elde edilmiş olduğu gözlenmektedir. Bu yöntemde de genişletme süresi, sıkıştırma süresinin yaklaşık **1,2** katı daha fazladır (GS = 1,2 * SS).

5.6.3 KrDDH, IHeDSH ve KeDDH yöntemlerinin karşılaştırılması

Kelimeye Dayalı Dinamik Huffman yönteminden elde edilen sıkıştırma verimi, karaktere göre sıkıştırmada en iyi sonucu veren *Karaktere Dayalı Dinamik Huffman*, sembol sıkıştırmada en iyi sonucu veren *İlk Heceye Dayalı Statik Huffman* yönteminden alınmış olan sıkıştırma verimleri ile karşılaştırılmalı olarak Çizelge 5.14'de verilmekte olup, Şekil 5.21'de de grafik olarak gösterilmektedir.

Çizelge 5.14 KrDDH ile IHeDSH yönteminin, KeDDH yöntemi ile karşılaştırılması

	KrDDH			IHeDDSH			KeDDH		
Sıra No	**η %**	**SS (ms)**	**SS/GS (ms)**	**η %**	**SS (ms)**	**SS/GS (ms)**	**η %**	**SS (ms)**	**SS/GS (ms)**
1	**40,0**	170	0,26	**46,8**	280	0,32	**44,6**	2.030	0,84
2	**40,4**	220	0,21	**47,5**	490	0,57	**47,0**	2.800	0,82
3	**38,1**	220	0,18	**43,6**	600	0,39	**45,3**	3.790	0,85
4	**40,1**	280	0,20	**46,6**	710	0,41	**48,5**	4.170	0,83
5	**39,9**	490	0,18	**45,3**	1.430	0,42	**49,3**	9.060	0,86
6	**39,3**	720	0,18	**41,1**	2.150	0,43	**53,7**	4.780	0,74
7	**39,9**	880	0,20	**45,3**	2.360	0,40	**51,4**	10.270	0,82
8	**40,6**	1.260	0,19	**46,2**	3.630	0,42	**51,6**	22.850	0,86
9	**39,6**	1.370	0,19	**44,2**	3.730	0,40	**51,1**	25.100	0,86
10	**46,2**	3.300	0,18	**54,2**	8.070	0,39	**58,1**	39.270	0,84
11	**51,4**	3.360	0,19	**59,5**	7.070	0,35	**61,1**	28.460	0,77
12	**46,9**	5.280	0,20	**55,3**	12.360	0,39	**59,2**	58.270	0,84
13	**46,1**	490	0,18	**54,0**	22.460	0,40	**58,9**	77.880	0,75
14	**43,9**	11.580	0,20	**50,6**	28.450	0,40	**56,5**	131.230	0,79



Şekil 5.21 Gerçek metin dosyası ile KrDDH, IHeDSH ve KeDDH'dan alınan sonuçların karşılaştırılması

Karşılaştırma sonucunda, *Kelimeye Dayalı Dinamik Huffman* yöntemi sıkıştırma veriminde, *İlk Heceye Dayalı Statik Huffman* yöntemine göre ortalama **%4**, *Karaktere Dayalı Dinamik Huffman* yöntemine göre de ortalama **%12**'lik daha iyi bir sonuç vermiştir. Büyük metinlerde *Kelimeye Dayalı Dinamik Huffman* yöntemi ile, *İlk Heceye Dayalı Statik Huffman* yöntemleri arasında ortalama **%5**'e varan bir başarı artışı gözlenmektedir. Küçük metinlerde başarının düşük olmasının sebebi ise, her iki yöntemde de ekler aynı kaldığı halde ilk hece ve köklerde farklılıkların olmasıdır. Hece uzunluğunu 3 karakter kabul ettiğimizde, kök-gövde uzunluğunun da daha fazla karakter ile ifade edilmesine karşılık, sayıca hecelerden daha fazla olduğundan Huffman kodlama ağacında daha fazla seviye oluşmaktadır. Buda kök ve gövdelere daha uzun kodların gelmesine neden olmaktadır.

Sıkıştırma ve genişletme süreleri en az süren yöntem *Karaktere Dayalı Dinamik Huffman* yöntemidir. Sıkıştırma ve genişletme süreleri arasında $\mathbf{GS_{KrDDH} = 5{,}2 * SS_{KrDDH}}$ bağıntısı vardır. Bunun sebebi metnin harf harf kodlanmasıdır. *Kelimeye Dayalı Dinamik Huffman* ile *İlk Hece Dayalı Statik Huffman* yöntemlerinde kelimenin kök-gövde/hece ve eklerine ayrılma işlemleri söz konusu olduğundan sıkıştırma ve genişletme süreleri daha uzun sürmektedir. Bu iki yöntemin sıkıştırma ve genişletme süreleri arasında da 5.8'deki bağıntılar mevcuttur.

$$GS_{IHeDSH} = 2{,}5 * SS_{IHeDSH}$$

$$GS_{KeDDH} = 1{,}2 * SS_{KeDDH} \qquad (5.8)$$

5.7 Geliştirilmiş Kelimeye Dayalı Dinamik Huffman Yöntemi

Bu çalışma kapsamında gerçekleştirilen *Kelimeye Dayalı Dinamik Huffman* yöntemi ile elde edilmiş başarıyı daha da iyileştirebilmek ve sıklıkların normalizasyonu sırasında karşılaşılan kayıpları ortadan kaldırmak amacıyla yeni bir yöntem geliştirilmiştir. Geliştirilen bu yöntem ile kod çözücünün sıkıştırılmış veriyi açmak üzere kullanacağı Huffman ağacını oluşturmak için ihtiyaç duyduğu sıklık bilgileri yerine, kullanılmış olan Huffman ağacının yapısı doğrudan aktarılmaktadır.

5.7.1 "Ağaç için özel kod bilgisi" yöntemi

"Ağaç için özel kod bilgisi" adını vereceğimiz bu yöntem, aşağıdaki avantajları sağlamaktadır.

1. Ağaç oluşturmak için gerekli alanı 1/5 ile 1/10 seviyesinde azalmaktadır.
2. Kullanım sıklığı bilgilerini doğrudan gönderdiğimizde, sıkıştırma işlemindeki sıklık bilgilerinin bit uzunluğunun artması, sıkıştırma verimini arttırmakta buna karşılık başlık verimini düşürmektedir. Yani başlık verimi ile sıkıştırma verimi ters orantılıdır.

$$(\eta_{başlık} * \eta_{sıkıştırma}) = c_{sabit}$$

Oysa özel başlık kodu kullanılması ile bu çelişkili durum ortadan kalkmaktadır. Bu durumda başlık kodu, *N:kök sayısı*, *M:ek sayısı*, *bb: başlık boyu* olmak üzere,

$$bb = 2*(N+M)-4$$

gibi sabit değerde tutulabildiği için normalizasyonda aralık daraltmanın sebep olduğu sıkıştırma verimindeki düşüş ortadan kalkmaktadır. Kullanım sıklık bilgisine gerek kalmadan, oluşturulmuş olan Huffman ağacının gönderilmesiyle başlık bölümünün uzunluğu ortalama **%40** daha kısaltılmış, sıkıştırma verimindeki başarı da ortalama **%1,5** daha arttırılmış olup, kod çözülürken geçen sürede azaltılmıştır. Bu yöntem ile sıkıştırılmış dosyanın yeniden açılabilmesi için sıklık bilgilerinden yararlanarak Huffman kod ağacının yeniden oluşturulmasına gerek kalmamıştır. Sembollerin kodları sadece gönderilen ağaç bilgisinden elde edilmektedir. Başlık bilgisi içerisinde bulunan KGSB ve ESB alanlarına yazılan bit katarından yararlanarak Huffman ağacını kurmadan sembollerin kodlarının oluşturulması Bölüm 5.7.2'de anlatılmıştır.

Bir önceki bölümde sıklık bilgisinin taşınabilmesi için $\lceil 11 *(\text{kök-gövde sayısı} + \text{ek sayısı}) \rceil$ bite ihtiyaç olacağı belirtilmişti. Geliştirilen bu yöntem ise, n adet yaprağı olan bir Huffman

ağacı için en fazla (2n-2) adet bitin kullanılmasını gerektirmektedir. Sayısal bir örnek ile açıklayacak olursak;

Test-kümesi içerisinde yer alan "ecolasan.txt" dosyasında,

N=3994 M=414 olarak bulunmuştur.
KeDDH (*normalizasyon yapıldığında*) yönteminde başlık bölümünde sıklık bilgisini göndermek için,

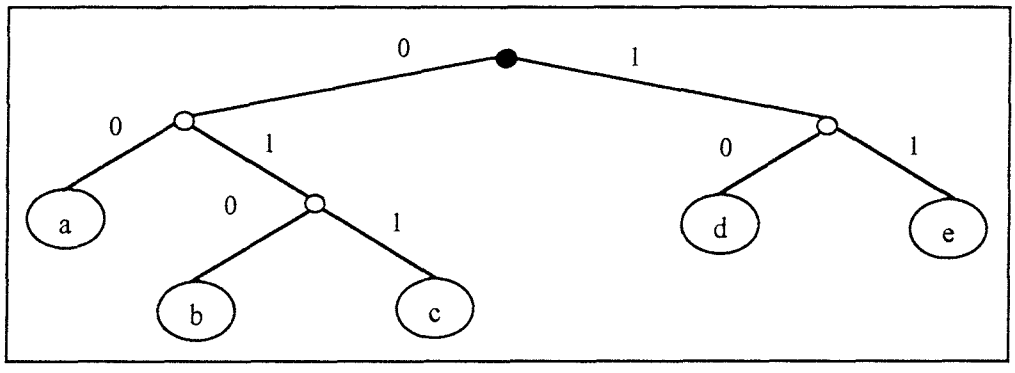
(N+M)*11 = 48.488 bit kullanılır iken,
GKeDDH yönteminde sadece,
2* (N+M)-4 = 8.812 bit kullanmak yeterli olacaktır.

Başlık bilgisi içerisinde yer alan kullanım sıklığı bilgisinin gönderilmesi için gerekli olan alan, sıklık bilgisine ait Huffman kodlama ağacının gönderilmesiyle ortalama **5,5** katı azalmıştır.

Huffman kodlama ağacının kod çözücüye gönderilmesinde, ağacın yön bilgilerinden yararlanılmıştır. Yön bilgisi "0" ve "1" değerine sahip olup bir bit ile ifade edilmektedir. Uygulamada "0" değeri ağaç üzerinde sola giderken bir seviye aşağı inmeyi, "1" değeri ise ağaç üzerinde sağa doğru giderken bir seviye yukarı çıkmayı sağlamaktadır. Yön bilgisini gösteren 0 ve 1 değerleri gelme sıralarına göre başlık bölümündeki KGSB/ESB alanında, ağacın dallarındaki yapraklara denk düşen değerler ise KGAB/EAB alanında tutulmaktadır. Hem kodlama hem de kod çözme işleminin tutarlı çalışabilmesi için yön bilgilerinin kodlanması her zaman soldan sağa doğru gerçekleştirilmelidir. Bu şekilde L-N-R (*left-node-right*) adını verdiğimiz "soltaraf-düğüm-sağtaraf" notasyonu kullanılarak, bir ağacın kökünden başlayıp sola doğru en alt yaprağına kadar gidilmekte, daha sonra bir üst seviyeye çıkılıp sağ tarafa dallanılmaktadır. Bulunulan yerin yaprak olup olmadığı kontrol edilip, varılan düğüm noktası yaprak değilse aynı işlem tekrar edilir. Eğer yaprak ise işlenmemiş bir sağ dal bulununcaya kadar bir üst seviyeye çıkılır. Kullanım sıklık bilgileri yerine başlık için oluşturulan özel kod katarı, kod çözücü tarafından yön bilgisi yardımıyla çözüldüğünde kök veya eke karşı gelen kodlar kolaylıkla elde edilmektedir.

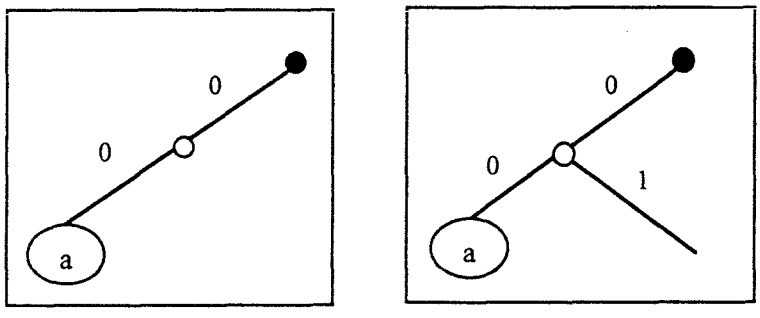
5.7.2 Sıklık bilgileri yerine gönderilen Huffman ağaç yön bilgisinin oluşturulması

Kodlayıcı, başlık bölümünde sıklık bilgisi yerine kurmuş olduğu ağaç bilgisini gönderirken aşağıdaki adımları izler. Verilen bu örneğin ekler için oluşturulmuş olduğunu kabul edip Şekil 5.22'deki ağacı örnek olarak alalım.



Şekil 5.22 Örnek Huffman ağacı

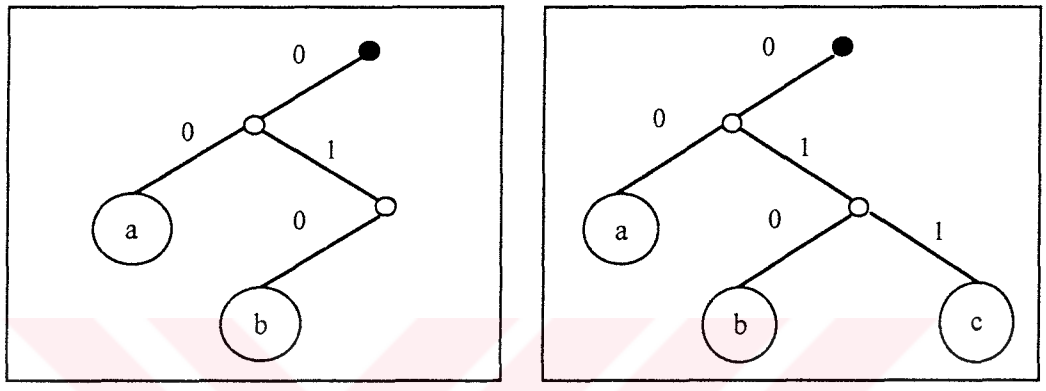
- İşleme kök (*root*) den başlanarak (*soltaraf-düğüm-sağtaraf*) L-N-R notasyonuna bağlı kalınıp ESB alanına "0" değeri konulur ve bir seviye aşağıya inilir.
- Gelinen düğüm bir yaprak değilse tekrar sol tarafa gidilir ve ESB alanına ikinci "0" değeri konulur ve bir seviye aşağıya inilir.
- Bulunulan düğüm noktası eğer bir yaprak ise EAB alanına (a) sembolünün adresi yerleştirilir (Şekil 5.23 a) ve bir üst düğüme geçilip LNR notasyonu gereği sağ tarafa dallanılır ve ESB alanına "1" değeri konulur ve bir seviye aşağıya inilir (Şekil 5.23 b).



(a) (b)

Şekil 5.23 (a) ESB alanına (00), EAB alanına da (a) sembolünün adresi yazıldı
(b) ESB alanında (001), EAB alanına da (a) sembolünün adresi yazıldı

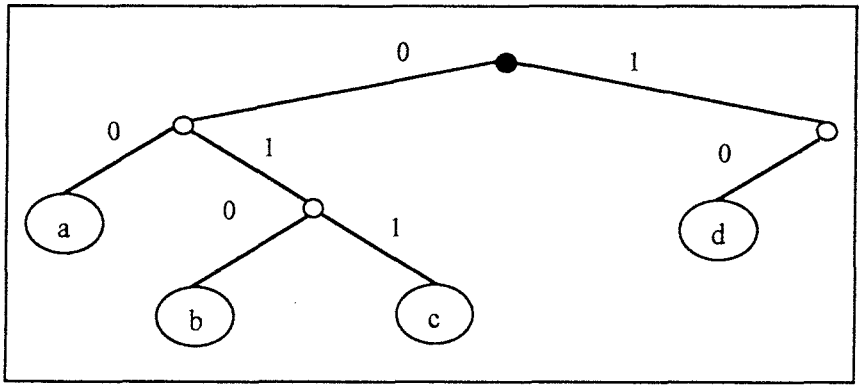
- Gelinen düğüm noktası bir yaprak olmadığından sol tarafa yönlenilir ve ESB alanına "0" değeri yazılır ve bir alt seviyeye inilir. Geldiğimiz düğüm noktası yaprak olduğundan EAB alanına (b) sembolünün adresi yazılır (Şekil 5.24 a) ve bir üst düğüme geri dönülüp, sağ tarafta bir dal varsa sağa yönlenilir yoksa sırasıyla üst düğümlerde boş sağ bir dal aranır.
- Bulunduğumuz düğümün sağ dalı mevcut olduğundan ESB alanına "1" değeri konulur ve yaprak olduğundan EAB alanına (c) sembolünün adresi yazılır (Şekil 5.28 b).



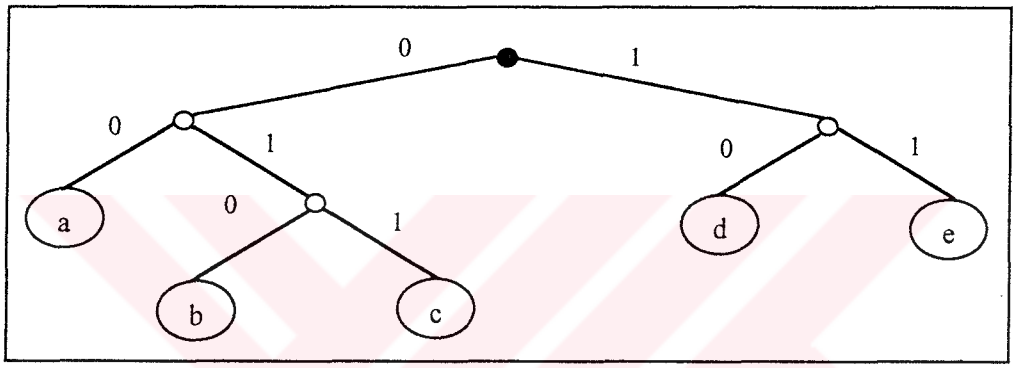
Şekil 5.24 (a) ESB alanına (0010), EAB alanına da (ab) sembollerinin adresleri yazıldı

(b) ESB alanına (00101), EAB alanına da (abc) sembollerinin adresleri yazıldı

- Sağ dalı işlenmemiş bir düğüm bulana kadar bir üst seviyeye çıkılıp işleme devam edilmelidir. Köke ulaştığımızda, ESB alanına "1" değerini koyup işleme tekrar L-N-R notasyonu gereği sol tarafa yönlenip bir yaprak değerine ulaşıncaya kadar devam edilir. Kökten sağa dallanıp bir seviye aşağıya indiğimizde geldiğimiz düğüm noktası yaprak değilse, sola dallanıp ESB alanına "0" değeri konulup, bir seviye aşağıya inilir. Gelinen düğüm noktasının yaprak olduğu anlaşılınca EAB alanına (d) sembolünün adresi yerleştirilip (Şekil 5.25) bir üst seviyeye çıkılarak sağ tarafa dallanılır.
- Sağ tarafa dallanıp gelinen düğümün yaprak olduğu anlaşılınca ESB alanına "1", EAB alanına da (e) sembolünün adresi yerleştirilir (Şekil 5.26).
- İşlem sonunda Şekil 5.26'daki ağaç elde edilmiş olur.



Şekil 5.25 ESB alanına (0010110), EAB alanına da (abcd) sembollerinin adresleri yazıldı



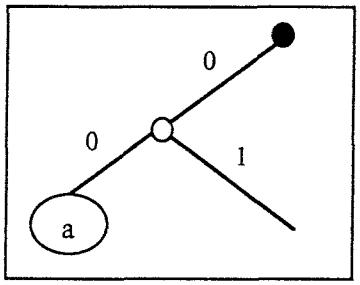
Şekil 5.26 ESB alanına (00101101), EAB alanına da (abcde) sembollerinin adresleri yazıldı

"Başlık" bölümüne gönderilecek olan bu bit katarının oluşturulması için gerekli olan algoritmanın akış diyagramı EK 2 'de verilmiştir.

ESB (*Ek sıklık bilgisi*) **: (00101101)**
EAB (Ek adres bilgisi) **: (abcde)** sembollerinin adres değerlerine sahip olacaktır.

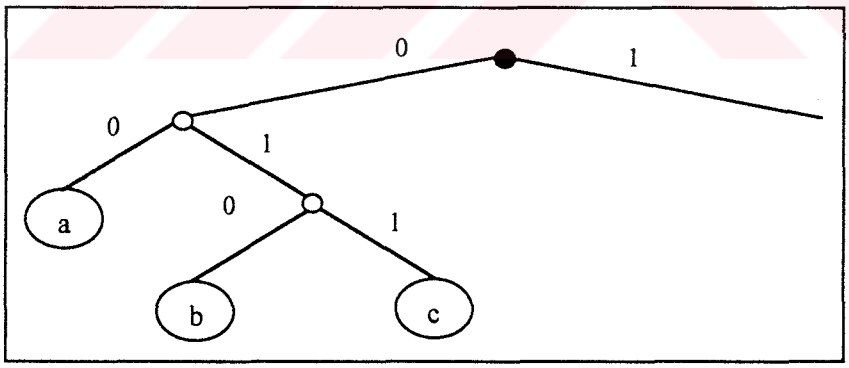
Kod çözücü, sıkıştırılmış dosyayı açabilmek için başlık bilgisini çözmeyle işe başlar. Sıklık bilgisi yerine gönderilen ağaç bilgisini çözerken izlenecek yol aşağıda anlatılmaktadır.

- ESB alanından aldığımız "0" değeri ile kökün sol tarafına bir dal çizilip bir seviye aşağıya inilir, aldığımız ikinci "0" değeri ile bulunduğumuz düğümün sol tarafına bir dal daha çizip, okuduğumuz üçüncü değerin "1" olduğunu anladığımızda EAB alanından ilk sembolü çekip, bir üst seviyeye çıkıp sağ tarafa bir dal ekleriz (Şekil 5.27).



Şekil 5.27 ESB alanından (001), EAB alanından da (a) sembolü çekildi

- Gelen üçüncü "0" değeri ile bulunduğumuz düğümün sol tarafına yeni bir dal ekleyip, ikinci "1" değeri ile EAB alanından sıradaki (b) sembolünü çekip, indiğimiz düğümün sağ tarafına bir dal ekleriz. Gelen üçüncü "1" değeri, EAB alanından (c) sembolünü çekmemizi ve sağ tarafı işlem görmemiş bir düğüm aramamız gerektiğini söyler ve buda bizi bu örnek için köke (*root*) çıkarıp sağ tarafa bir dal eklememizi ve bir seviye aşağıya inmemizi gerektirir (Şekil 5.28).
- Gelen dördüncü "0" değeri ile bulunulan düğümün sol tarafına bir dal eklenir ve son "1" değeri ile EAB alanından (d) değeri çekilir, mevcut düğümün sağ tarafına bir dal eklenerek, EAB alanından son (e) değeride çekilerek Şekil 5.22'deki Huffman ağacı elde edilmiş olur.



Şekil 5.28 ESB alanından (001011), EAB alanından da (abc) sembolleri çekildi

Başlık bölümünde gönderilen bit katarının çözülerek her bir sembolün kodunun üretilmesi için gerekli olan algoritmanın akış diyagramı Ek 3'de verilmiştir.

Başlık bölümünü yukarıda anlattığımız yöntem ile kodladığımızda *Geliştirilmiş Kelimeye Dayalı Dinamik Huffman* yönteminden alınan değerler Çizelge 5.15'de, grafik gösterilimide Şekil 5.29'da sunulmuştur.

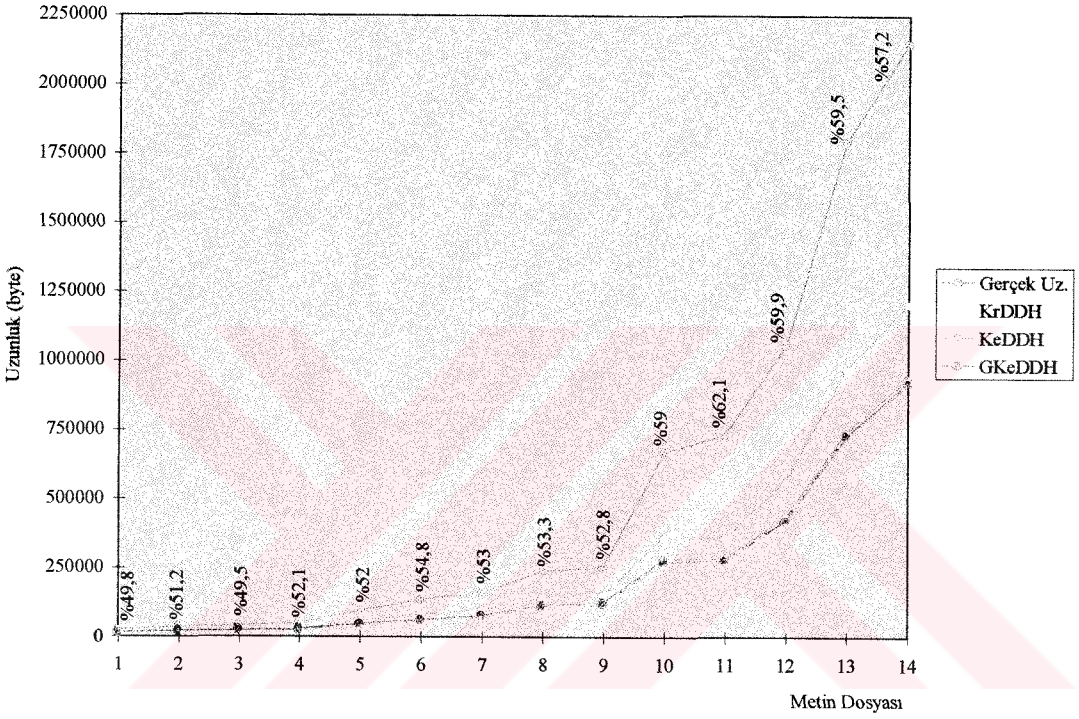
Çizelge 5.15 Gerçek metin dosyası ile GKeDDH yönteminden alınan sonuçların karşılaştırılması

Sıra No	Dosya Adı	Gerçek Uzunluk	GKeDDH	SS (ms)	η%	GS (ms)
1	Donat.txt	24.624	12.365	2.910	49,8	930
2	Gciva.txt	36.596	17.877	4.450	51,2	1.370
3	Takyol.txt	41.377	20.897	5.770	49,5	1.650
4	Masik.txt	48.881	23.393	6.370	52,1	1.870
5	Tubitak.txt	93.884	45.085	14.060	52,0	3.510
6	Phdthesis.txt	135.361	61.213	6.150	54,8	4.450
7	Ykemal.txt	160.240	75.375	15.100	53,0	5.870
8	Furuzan.txt	236.705	110.587	36.740	53,3	8.680
9	Omersey.txt	251.606	118.851	40.150	52,8	9.440
10	Oeksi.txt	666.120	273.203	56.360	59,0	22.030
11	Heper.txt	731.929	277.309	39.930	62,1	21.860
12	Ecolasan.txt	1.055.244	422.981	80.420	59,9	33.620
13	Mabirand.txt	1.795.226	727.479	106.770	59,5	59.810
14	Caltan.txt	2.147.095	918.945	193.620	57,2	80.910

Geliştirilmiş Kelimeye Dayalı Dinamik Huffman yöntemi, sıkıştırma veriminde ortalama **%58**'lik bir başarı sağlamıştır. Başlık bölümünde kullanım sıklığı bilgisi yerine, Huffman ağacının bilgisi gönderildiğinden genişletme bölümünde Huffman ağacının yeniden oluşturulmasına gerek kalmamıştır. Bu durum genişletme süresinin azalmasına neden olmuştur. *Kelimeye Dayalı Dinamik Huffman* yönteminde genişletme süresi, sıkıştırma süresinin 1,2 katı iken (5.8) *Geliştirilmiş Kelimeye Dayalı Dinamik Huffman* yönteminde ise genişletme süresi, sıkıştırma süresinden daha küçük olup aralarında (5.9)'daki gibi bir bağıntı vardır.

$$GS_{GKeDDH} = SS_{GKeDDH} / 2,9 \qquad (5.9)$$

Geliştirilmiş Kelimeye Dayalı Dinamik Huffman yöntemi, *Kelimeye Dayalı Dinamik Huffman* yöntemine göre sıkıştırma veriminde ortalama **%1**'lik, *Karaktere Dayalı Dinamik Huffman* yönteminde de **%13**'lük daha iyi bir başarı elde etmiştir.



Şekil 5.29 Gerçek metin dosyası ile KrDDH, KeDDH ve GKeDDH yönteminin sıkıştırma verimlerinin karşılaştırılması

Başlık bölümünde yapılan bu iyileştirme ile *Kelimeye Dayalı Dinamik Huffman* yönteminin genişletme süresi azaltılmış ve aynı zamanda sıkıştırma verimi arttırılmıştır. Bu iki yöntemin sıkıştırma ve geliştirme süreleri karşılaştırmalı olarak Çizelge 5.16'da verilmekte olup, bu süreler arasında (5.10)'daki gibi bağıntılar mevcuttur.

$$SS_{GKeDDH} = 1,45 * SS_{KeDDH}$$

$$GS_{GKeDDH} = 0,5 * GS_{KeDDH} \qquad (5.10)$$

Çizelge 5.16 GKeDDH ile KeDDH yöntemlerinin sıkıştırma ve genişletme sürelerinin karşılaştırılması

	KeDDH			GKeDDH		
Sıra No	η %	SS (ms)	SS/GS (ms)	η %	SS (ms)	SS/GS (ms)
1	**44,6**	2.030	0,84	**49,8**	2.910	3,12
2	**47,0**	2.800	0,82	**51,2**	4.450	3,24
3	**45,3**	3.790	0,85	**49,5**	5.770	3,50
4	**48,5**	4.170	0,83	**52,1**	6.370	3,41
5	**49,3**	9.060	0,86	**52,0**	14.060	4,00
6	**53,7**	4.780	0,74	**54,8**	6.150	1,37
7	**51,4**	10.270	0,82	**53,0**	15.100	2,57
8	**51,6**	22.850	0,86	**53,3**	36.740	4,23
9	**51,1**	25.100	0,86	**52,8**	40.150	4,25
10	**58,1**	39.270	0,84	**59,0**	56.360	2,56
11	**61,1**	28.460	0,77	**62,1**	39.930	1,83
12	**59,2**	58.270	0,84	**59,9**	80.420	2,40
13	**58,9**	77.880	0,75	**59,5**	106.770	1,79
14	**56,5**	131.230	0,79	**57,2**	193.620	2,39

5.7.3 GKeDDH yönteminin ikinci kez sıkıştırılması (ÖYT)

Geliştirilmiş Kelimeye Dayalı Dinamik Huffman yöntemi ile sıkıştırılmış bir metin dosyasına Örüntü Yerdeğiştirme (*Patern Subsition*) yöntemiyle ikinci kez sıkıştırma uyguladığımızda büyük metinlerdeki başarı ortalama % 0,5 artmaktadır. Buna karşılık küçük metinlerde hiç sıkıştırma yapılamamaktadır. *Geliştirilmiş Kelimeye Dayalı Dinamik* Huffman yöntemiyle sıkıştırılan dosyalara ikinci kez sıkıştırma uygulandığında alınan sonuç değerleri Çizelge 5.17'de gösterilmiştir.

Örüntü yereğiştirme yöntemi, veri içerisinde çok sık kullanılan kelimelerin veya bit desenlerinin yerine daha kısa kelimeler veya daha kısa bit desenleri koyma tekniğine dayanır. Burdan yola çıkılarak sıkıştırılmış dosya içerisinde en çok sıklıkla kullanılmış olan üçlü byte gurubu yerine, o dosya içerisinde kullanılmayan ikili byte gruplarından birinin yerleştirilmesi yoluna gidilmiştir.

Çizelge 5.17 GKeDDH yöntemi ile sıkıştırılmış dosyanın ÖYT ile ikinci kez sıkıştırıldığında alınan sonuçlar

Sıra No	Dosya Adı	Gerçek Uzunluk	GKeDDH	η%	ÖYT	SS (ms)	η%	GS (ms)
1	Donat.txt	24.624	12.365	49,8	12.359	50	49,8	< 1
2	Gciva.txt	36.596	17.877	51,2	17.850	60	51,2	< 1
3	Takyol.txt	41.377	20.897	49,5	20.899	50	49,5	< 1
4	Masik.txt	48.881	23.393	52,1	23.341	50	**52,2**	< 1
5	Tubitak.txt	93.884	45.085	52,0	45.074	160	52,0	< 1
6	Phdthesis.txt	135.361	61.213	54,8	61.213	170	54,8	50
7	Ykemal.txt	160.240	75.375	53,0	75.373	220	53,0	60
8	Furuzan.txt	236.705	110.587	53,3	110.571	330	53,3	60
9	Omersey.txt	251.606	118.851	52,8	118.845	390	52,8	60
10	Oeksi.txt	666.120	273.203	59,0	272.494	820	**59,1**	160
11	Heper.txt	731.929	277.309	62,1	270.805	820	**63,0**	160
12	Mabirand.txt	1.795.226	727.479	59,5	716.377	2.140	**60,1**	540
13	Ecolasan.txt	1.055.244	422.981	59,9	420.227	1.270	**60,2**	280
14	Caltan.txt	2.147.095	918.945	57,2	910.814	2.700	**57,6**	550

Örüntü Yerdeğiştirme Tekniğini kullanarak sıkıştırılmış bir dosyayı ikinci kez sıkıştırdığımızda, sıkıştırma verimi ortalama **%59** olmuştur. *Geliştirilmiş Kelimeye Dayalı Dinamik Huffman* yönteminde bu oran **%58** idi. Böylece **%1**'lik bir iyileşme sağlanmıştır. Küçük metinlerin sıkıştırma oranlarında hiçbir değişikliğin olmamasının sebebi bu metinlerde tekrar eden üçlü byte kombinasyonunun ikinci kez gelme olasılığının çok zayıf olmasıdır. Olasılığın artması, metnin uzunluğu ile doğru orantılıdır.

5.8 KrDSH, KrDDH, HeDSH, KökDSH, IHeDSH, KeDDH ve GKeDDH Yöntemlerinin Birbirleri ile Karşılaştırılması

Farklı uzunlukdaki dört adet metin dosyasının, sırası ile yedi farklı yöntem kullanarak sıkıştırılması sonucunda elde edilen sıkıştırma verimi, sıkıştırma süresi ve sıkıştırma süresinin genişletme süresine oranı Çizelge 5.18'de karşılaştırmalı olarak verilmiştir.

Çizelge 5.18 Yedi farklı yöntemin birbiri ile karşılaştırılması

	Omersey.txt (251.606 byte)			Oeksi.txt (666.120 byte)			Ecolasan.txt (1.055.244 byte)			Caltan.txt (2.147.095 byte)		
	η	SS(ms)	SS/GS	η	SS(ms)	SS/GS	η	SS(ms)	SS/GS	η	SS(ms)	SS/GS
KrDSH	38,7	1.210	0,16	45,9	3.020	0,17	46,7	4.670	0,16	43,9	10.000	0,16
KrDDH	39,6	1.370	0,19	46,2	3.300	0,18	46,9	5.280	0,20	43,9	11.580	0,20
HeDSH	37,0	4.560	0,37	49,8	9.880	0,38	50,8	15.210	0,37	43,8	34.870	0,37
KökDSH	43,7	2.900	0,32	53,1	8.460	0,42	54,2	12.960	0,42	49,9	29.770	0,43
IHeDSH	44,2	3.730	0,40	54,2	8.070	0,39	55,3	12.360	0,39	50,6	28.450	0,40
KeDDH	51,1	25.100	0,86	58,1	39.270	0,84	59,2	58.270	0,84	56,5	131.230	0,79
GKeDDH	52,8	40.150	4,25	59,0	56.360	2,56	59,9	80.420	2,40	57,2	193.620	2,39

KrDDH yöntemi, KrDSH yöntemine göre sıkıştırma veriminde daha iyi sonuç vermiş olup, her iki yöntemin sıkıştırma verimindeki başarısı ortalama %45'dir. KrDDH yönteminin sıkıştırma süresi, KrDSH yöntemine göre daha uzundur. KrDDH yönteminde, metin içerisinde kullanılan karakterlerin Huffman kodlarının oluşturulması aynı algoritma içerisinde yer aldığından sıkıştırma süresi artmaktadır. Bu iki yöntemde sıkıştırma ve genişletme sürelerinin birbirlerine göre oranı,
$SS_{KrDSH} / GS_{KrDSH} = 0,18$ ve $SS_{KrDDH} / GS_{KrDDH} = 0,2$ 'dir.

HeDSH yönteminde, sıkıştırma verimi ortalama %47 olup, karaktere dayalı yöntemlere göre %2'lik bir artış göstermiştir. Bazı metin dosyalarında bu başarı düşmekte (Omersey.txt dosyası) ve başarının düşük kaldığı metin dosyalarında kullanılan hecelerin, şablon içerisindeki kullanım sıklıklarının az olması dolayısıyla Huffman kodunun daha fazla bit ile ifade edilmesi gerekmekte ve bu da başarıyı düşürmektedir. Sıkıştırma süresi metin dosyasının uzunluğuna bağlı olarak artmakta olup sıkıştırma süresinin, genişletme süresine oranı $SS_{HeDSH} / GS_{HeDSH} = 0,4$'dür.

KökDSH yönteminde, sıkıştırma verimi ortalama %52 olup, karaktere dayalı yöntemlere göre sıkıştırma veriminde ortalama %7'lik, heceye dayalı yöntemlere göre de %5'lik bir artış göstermiştir. Sıkıştırma süresi metin dosyasının uzunluğu ile doğru orantılı olup, HeDSH yöntemine göre daha kısa sürmektedir. Sıkıştırma süresinin, genişletme süresine oranı ise $SS_{KökDSH} / GS_{KökDSH} = 0,41$'dir.

Statik yöntemlerin sonuncusu olan IHeDSH yönteminden alınan sıkıştırma verimi ortalama %53'dür. Sıkıştırma süresi HeDSH ve KökDSH yöntemlerine göre daha kısa sürmektedir. Bu yöntem yukarıda bahsedilmiş olan dört yönteme göre sıkıştırma veriminde en iyi sonucu vermiştir. Kendi içerisindeki başarı, diğer bütün yöntemlerde olduğu gibi sıkıştırılacak metin dosyasının uzunluğuna değil, içeriğine bağlıdır. IHeDSH yöntemi, karaktere dayalı yöntemlere göre %8, KökDSH yöntemine göre de %1'lik bir başarı sağlamıştır. Sıkıştırma süresinin, genişletme süresine oranı ise $SS_{IHeDSH} / GS_{IHeDSH} = 0,4$'dür.

Dinamik yöntemlerden olan *Kelimeye Dayalı Dinamik Huffman* yöntemi sıkıştırma veriminde ortalama %57'lik bir başarı sağlamıştır. Sıkıştırma süresi diğer yöntemlere göre

daha fazla olup, sebeplerinden biri, statik yöntemlerde önceden hazırlanmış şablonların kullanılması yerine, sıkıştırılacak metin dosyasının istatiksel analizi yapılarak karakter/sembollere özel Huffman kodlarının verilmesi; bir diğer sebep ise kök/gövde ve ekler için oluşturulmuş olan iki farklı sözlük içerisinden arama işleminin gerçekleştirilmesidir. KeDDH yöntemi, karaktere dayalı yöntemlere göre sıkıştırma veriminde ortalama %12, IHeDSH yönteminden ise ortalama %4'lük daha iyi bir başarı sağlamıştır. Sıkıştırma süresinin, genişletme süresine oranı SS_{KeDDH} / GS_{KeDDH} = 0,8'dir.

Son yöntem olan GKeDDH yöntemi, sıkıştırma veriminde ortalama %58'lik bir başarı elde etmiştir. Sıkıştırılmış dosyasının ÖYT tekniği kullanılarak ikinci kez sıkıştırılması ile sıkıştırma verimi %59'a çıkmıştır. Bu yöntem, diğer yöntemlere göre daha iyi sonuç vermiş olup, büyük metin dosyalarında sıkıştırma verimindeki başarı %60'lara ulaşmıştır. Başarının artışı, diğer altı yöntemde olduğu gibi metnin içeriğine bağlı olarak değişmektedir.

GKeDDH yöntemi (ÖYT uygulanmış), karaktere dayalı yöntemlere göre sıkıştırma veriminde %14'lük, *Kelimeye Dayalı Dinamik Huffman* yöntemine göre de %2'lik bir artış sağlamıştır. Başlık bilgisinde gerçekleştirilen bu iyileşme küçük dosyalarda sıkıştırma veriminin, büyük dosyalara oranla daha iyi sonuç vermesine neden olmuştur. Küçük dosyalarda kök-gövde ve eklerin kullanım sıklıklarının daha az olamasına karşılık, sıklık bilgisini göndermek için büyük metin dosyalarında olduğu gibi 11 bit kullanılmaktaydı. Geliştirilen bu yöntem ile gereksiz bit kullanımının önüne geçilmiş, n elemana ait sıklık bilgisinin gönderilmesi yerine huffman ağacının kendisi gönderilerek, n eleman için (2n-2) adet bit kullanılmıştır.

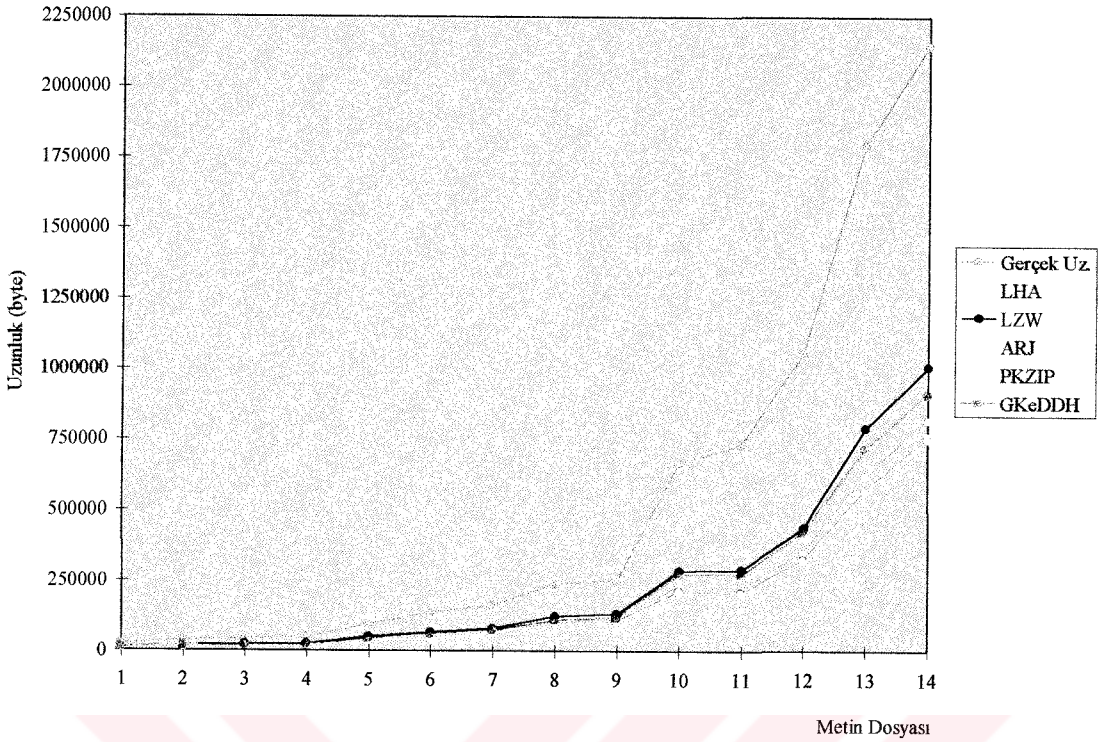
İbranice ve İngilizce metin dosyalarında önek ve sonekleri kullanarak gerçekleştirilmiş sıkıştırma işleminde elde edilen başarı İbranice dili için %37, İngilizce dili için %45 olarak bulunmuştur (A.S.Fraenkel vd., 1989). Türkçe'nin biçimbilimine dayalı gerçekleştirilen bu yöntem ile sıkıştırma veriminde %60-65'lere varan başarı elde edilmiştir. Başarıyı arttırmak için konuya özel sözlükler oluşturularak %70'lere varan bir sıkıştırma veriminin alınabileceği tahmin edilmektedir.

5.9 Geliştirilen Kelimeye Dayalı Dinamik Huffman Yönteminin Diğer Yöntemler ile Karşılaştırılması

Geliştirilen Kelimeye Dayalı Dinamik Huffman yöntemi ile Çizelge 5.17'de verilen sözlük kullanan yöntemlerden LZW ile karşılaştırma yapıldığında ortalama %4'lük daha iyi bir sıkıştırma verimi elde edilmiştir. ARJ ve PKZIP yöntemleri ise sıkıştırma veriminde ortalama %8 oranında daha iyi sonuç vermektedir. *Kelimeye Dayalı Dinamik Huffman* yönteminin diğer yöntemlerle karşılaştırılması Çizelge 5.19'da verilmekte olup, grafik gösterilimi de Şekil 5.30'da sunulmuştur.

Çizelge 5.19 GKeDDH yönteminin diğer sözlük yöntemleri ile karşılaştırılması

Sıra no	Dosya Adı	LHA	η %	LZW	η %	ARJ	η %	PKZIP	η %	GKeDDH	η %
1	Donat.txt	10.038	59,2	12.429	49,5	10.025	59,3	9.946	59,6	12.359	49,8
2	Gciva.txt	13.223	63,9	18.575	49,2	13.013	64,4	12.922	64,7	17.850	51,2
3	Takyol.txt	17.289	58,2	22.998	44,4	16.905	59,1	16.859	59,3	20.899	49,5
4	Maşık.txt	16.766	65,7	25.848	47,1	16.373	66,5	16.235	66,8	23.341	52,2
5	Tubitak.txt	41.090	56,2	50.951	45,7	39.032	58,4	38.866	58,6	45.074	52,0
6	Phdthesis.txt	44.682	66,9	66.300	51,0	41.486	69,4	41.427	69,4	61.213	54,8
7	Ykemal.txt	65.690	59,0	81.005	49,4	61.810	61,4	61.454	61,6	75.373	53,0
8	Furuzan.txt	107.214	54,7	124.151	47,6	101.249	57,2	100.554	57,5	110.571	53,3
9	Omersey.txt	111.435	55,7	132.186	47,5	105.291	58,2	104.571	58,4	118.845	52,8
10	Oekşi.txt	228.365	65,7	285.230	57,2	211.594	68,2	209.474	68,6	272.494	59,1
11	Heper.txt	217.245	70,3	290.230	60,4	204.607	72,0	202.604	72,3	270.805	63,0
12	Mabirand.txt	622.714	65,3	791.867	55,9	570.557	68,2	565.183	68,5	716.377	60,1
13	Ecolasan.txt	348.765	66,9	437.021	58,6	323.880	69,3	320.504	69,6	420.227	60,2
14	Caltan.txt	817.869	61,9	1.009.863	53,0	760.005	64,6	751.531	65,0	910.814	57,6



Şekil 5.30 GKeDDH yönteminin sözlük kullanan diğer yöntemler ile karşılaştırılması

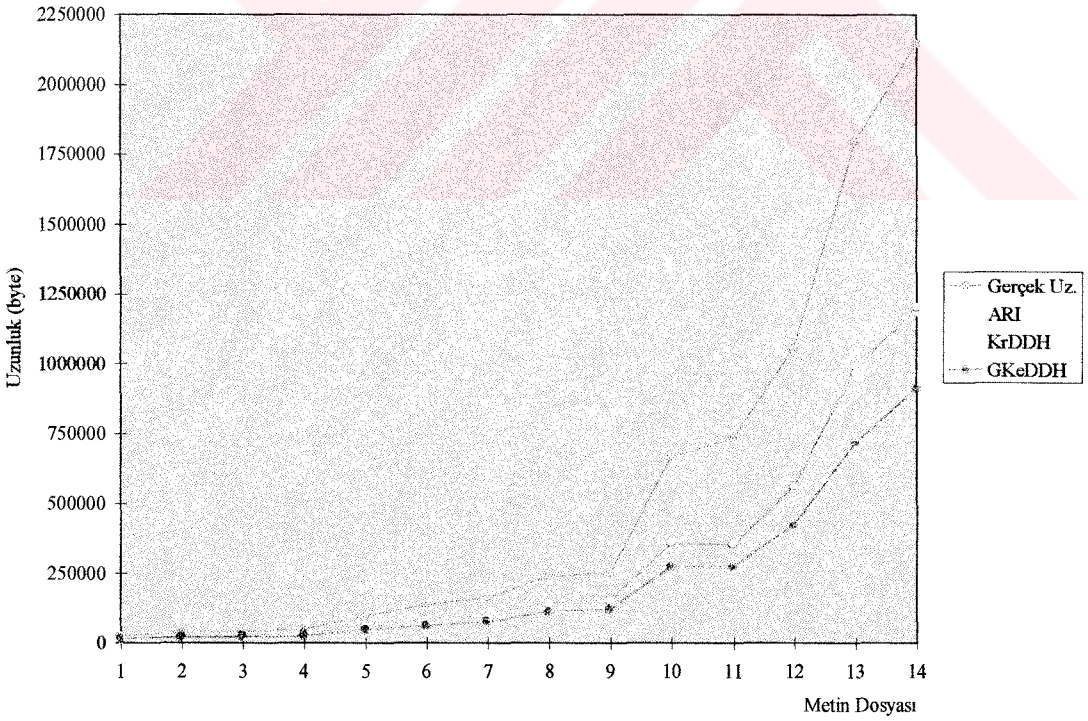
Kelimeye Dayalı Dinamik Huffman yöntemi, kendi alanındaki sıklığa bağlı yöntemler (ARI, Arithmetic Coding; *Karaktere Dayalı Dinamik Huffman*) ile karşılaştırıldığında sıkıştırma veriminde ortalama %14'lük bir artış izlenmiştir. *Geliştirilmiş Kelimeye Dayalı Dinamik Huffman* yönteminin sıklığa bağlı yöntemler ile karşılaştırılması Çizelge 5.20'de verilmiş olup, grafik gösterilimi de Şekil 5.31'de sunulmuştur.

5.10 Kullanım Sıklıklarının Matematiksel Modeli

Bütün sıkıştırma algoritmalarında olduğu gibi bu çalışmada da kullanılan algoritmaların sıkıştırma verimi, sıkıştırılan metnin içeriğine ve boyuna bağlı olarak değişimler göstermektedir. Verimin, metine bağımlılığını ortadan kaldırmak için orijinal metinde geçen kelimelerin kullanım sıklığını ve sıklık sırasına bağlayan matematiksel modeli test-küme'leri yardımıyla çıkarmaya çalışalım.

Çizelge 5.20 GKeDDH yönteminin sıklığa bağlı diğer sıkıştırma yöntemleriyle karşılaştırılması

Sıra no	Dosya Adı	ARI	η %	KrDDH	η %	GKeDDH	η %
1	Donat.txt	14.500	41,1	14.769	40,0	12.359	49,8
2	Gciva.txt	21.428	41,4	21.793	40,4	17.850	51,2
3	Takyol.txt	25.294	38,9	25.601	38,1	20.899	49,5
4	Maşık.txt	28.922	40,8	29.297	40,1	23.341	52,2
5	Tubitak.txt	56.030	40,3	56.456	39,9	45.074	52,0
6	Phdtheis.txt	81.374	39,9	82.145	39,3	61.213	54,8
7	Ykemal.txt	95.838	40,2	96.249	39,9	75.373	53,0
8	Furuzan.txt	140.243	40,8	140.653	40,6	110.571	53,3
9	Omersey.txt	151.441	39,8	151.999	39,6	118.845	52,8
10	Oekşi.txt	356.497	46,5	358.682	46,2	272.494	59,1
11	Heper.txt	350.997	52,0	355.987	51,4	270.805	63,0
12	Ecolasan.txt	554.929	47,4	560.838	46,9	420.227	60,2
13	Mabirand.txt	994.658	44,6	968.048	46,1	716.377	60,1
14	Caltan.txt	1.185.036	44,8	1.204.362	43,9	910.814	57,6



Şekil 5.31 GKeDDH yönteminin sıklığa bağlı sıkıştırma yöntemleri ile karşılaştırılması

Örneğin, *test-kümesi* içerisinde bulunan "takyol.txt" dosyasını incelediğimizde, herbir kod kelime uzunluğundan kaç adet kelime olduğu ve bunların toplam kullanım sıklıkları Çizelge 5.21'de verilmiştir.

Çizelge 5.21'in, 4.sütununda (134,73,32,...) olarak giden değişik kelimelerin kullanım sıklığının, kelime sayısına oranını (y), büyüklük kademelerini de (x) ile gösterirsek, y=f(x) fonksiyonunu bu değerler yardımıyla belirleyebiliriz. Bu fonksiyondaki (y) değerleri, (x) artarken azalmakta ve azalma geometrik olarak gitmekte olduğundan,

Çizelge 5.21 "takyol.txt" dosyasına ait istatistiki bilgi

Kod kelime uzunluğu	**Kullanım sıklığı**	**Kelime sayısı**	**Kullanım sıklığı / kelime sayısı (y)**	**Kademe (x)**
5	267	2	134	1
6	291	4	73	2
7	129	4	32	3
8	699	41	17	4
9	1010	108	9	5
10	8266	172	4	6
11	844	363	2	7
12	549	549	1	8
13	6	6	1	9

$$y = f(x) = \frac{a}{b^x} = ab^{-x} \qquad (5.11)$$

eşitik (5.11)'deki gibi olup, tarafların logaritması alınırsa,

$y=ab^{-x}$

$$\ln(y) = \ln(a) - (\ln b)x$$
$$\downarrow \quad \downarrow \quad \downarrow$$
$$Y = A + Bx$$

olup, Y=A+Bx doğrusu için *En Küçük Kareler* metodu uygulandığında denklem takımı,

$$\begin{bmatrix} n & \sum x \\ \sum x & \sum x^2 \end{bmatrix} * \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} \sum \ln(y) \\ \sum x\ln(y) \end{bmatrix}$$

şeklini alıp Çizelge 5.21'deki değerler için n=8 alınarak hesaplandığında,

$$\begin{bmatrix} 8 & 36 \\ 36 & 204 \end{bmatrix} * \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} 19.764 \\ 59.365 \end{bmatrix}$$

$A = 5.639 \qquad a=e^{A}=281.175$

$B = -0.704 \qquad b=e^{-B}=2.022$

$Y = 281.2 * 2.022^{-x}$

aranan büyüklük olup, (y) değerleri ile bu fonksiyondan elde edilen (Y) değerleri arasındaki mutlak hata Çizelge 5.22'de verilmiştir. Burada (5.11)'daki denklemde (b) değeri, b≅2 olup, *test-kümesi*'ndeki diğer dosyalarda da bu değer yaklaşık olarak aynı çıkmıştır.

Çizelge 5.22 En küçük kareler yönteminden alınan sonuç değerleri

x	x^2	y	ln(y)	xln(y)	$y=ab^{-x}$	$\lvert\varepsilon\rvert$
1	1	134	4.99	4.897	139.055	5.055
2	4	73	4.29	8.58	68.77	4.23
3	9	32	3.47	10.40	34.01	2.01
4	16	17	2.83	11.33	16.82	0.18
5	25	9	2.20	10.96	8.32	0.68
6	36	4	1.37	8.31	4.11	0.11
7	49	2	0.69	4.85	2.03	0.003
8	64	1	0	0	1.00	0.00
36	**204**	**272**	**19.763**	**59.365**		

"heper.txt" dosyası için, $y=1780 * 2^{-x}$ olup her iki değer için logaritmik eksen sistemine çizilecek grafiklerde eğimler aynı olmaktadır. Bu durumda *test-kümesi* dışında alınacak herhangibir metinde de (b) sabiti, $b\cong 2$ değerini gösterecektir. Bütün metinlerde en çok kullanılan kelimeleri aynı seviyeye getirmek üzere normalisazyon yapılırsa, $y=100*2^{-x}$ gibi bir denklem elde edilir ki bu modele uyan Huffman ağaçları yapısında farkedilebilir benzerlik görülmektedir.

Genetik Algoritmalar

Bu doktora çalışmasında doğal dillerin analizi yapılarak, bu analiz sonuçlarına göre sıkıştırma algoritması geliştirilmiştir. Seçilen bu modelde de *doğal*'lığa uygun olarak, matematik modelin geliştirilmesinde *En Küçük Kareler* yöntemi yerine daha doğal bir yöntem kullanmak üzere **Genetik Algoritmalar** (GA) seçilmiş ve *test-kümesi*'ndeki dosyalardan "donat.txt" ve "takyol.txt" üzerinde yapılan (GA) çalışmasıyla (5.11)'deki denklem yardımıyla (a) ve (b) katsayıları,

$a_{donat}\cong 82$ $a_{akyol}\cong 281$ $b_{donat}\cong b_{takyol}=2.034$ bulunup,

$y_{donat} = 82*2.034^{-x}$ $y_{akyol} = 281*2.034^{-x}$

olarak elde edildiği izlenmiştir.

6. SONUÇ

Veri sıkıştırma, bilgisayar sistemlerinde ve haberleşme hatlarında yer ve zaman kazancı sağlamak amacıyla, depolanacak veya iletilecek verinin kapladığı yeri, bilginin içeriğini bozmadan; uygulamaların özelliğine göre kayıpsız veya kayıplı olarak geri dönüşümü sağlayan yöntemlerle azaltmaya çalışması sebebiyle, bilişim dünyasındaki önemli araştırmalardan birini oluşturmaktadır. Yapılan bu doktora çalışmasında, Türkçe metinlerin biçimbilimsel olarak incelenmesiyle elde edilen istatistiksel verilere göre kelimelerin kök-gövde ve eklerini dinamik Huffman yöntemi ile kodlayan ve metni sıkıştıran bir sistem geliştirilmiştir.

Bu tez çalışmasının önemli ve diğer çalışmalara göre ayırt edici özelliği Türkçe metinlerin, Türk dilinin yapısına uygun şekilde analizinin yapılıp, elde edilen kök-gövde ve ek bilgilerinin kayıpsız geri dönüşümü sağlayacak şekilde sıkıştırılıp, açılmasını başarı ile gerçekleştirmiş olmasıdır. Konu ile ilgili bilimsel literatür incelendiğinde konuşma dillerine yönelik doğrudan biçimbilimsel özelliklerinden yararlanarak sıkıştırma yapan sistemlere rastlanmamıştır. Bu konuda (Fraenkel vd., 1983) tarafından İngilizce ve İbranice dillerindeki önek ve soneklerin incelenip sezgisel sıkıştırma yöntemleri önerilmiş olmasına rağmen, tam bir sıkıştırma ve açma yöntemi geliştirilmemiştir. Bugüne kadar veri sıkıştırma konusunda Türkçe ile ilgili biçimbilimsel analize dayalı sıkıştırma yöntemleri üzerinde çalışılmamış olması da bu konudaki açığı ortaya çıkartmaktadır.

Tez çalışması kapsamında yapılan araştırmalar ve incelemeler sırasında Türkçe bir metin içerisinde geçen karakterlerin kullanım sıklıklarının Zipf kanununa uygun davrandığının gösterilmesi de çalışmanın önemli noktalarından biridir. Türkçe metinlerin biçimbilimsel olarak kök-gövde ve eklerine ayrılması ve her metin için bu bilgilerin optimum uzunlukta bit ile gösterilecek şekilde, iki ayrı Huffman ağacına yerleştirilerek kodlanması önerilen sistemin dinamik yapıya sahip olmasını ve böylece sıkıştırılacak metnin yapısına en uygun çıkışı sağlamaktadır. Sıkıştırma ve açma işlemlerinin eş zamanlı yapılması gerekmediği için sıkıştırılan metinlerin önünde başlık adı verilen ve metin içerisinde kullanılan kök-gövde ve ekler ile geri dönüşüm işleminde kullanılacak Huffman ağaçlarının yapısını saklayan bir bilgi

bulunmaktadır. Başlığın uzunluğu özellikle kısa metinlerde sistem performansını doğrudan etkilemektedir. Bu nedenle bu doktora çalışmasında bir Huffman ağacının sahip olduğu elemanların sıklık bilgileri yerine, eleman başına iki biti geçmeyen bir maliyet ile ifade edilmesini sağlayan ve geri dönüşüm esnasında Huffman ağacı oluşturulmadan doğrudan elemanlara ait Huffman kod kelimelerinin elde edilmesini sağlayan bir yöntem geliştirilmiştir. Geliştirilen yöntem ile (n) elemanı olan bir ağaç için bu özel başlık bilgisi, (2n-2) adet bit ile ifade edilmektedir. Sistemin sonlandırılması için geçersiz bir bit kombinasyonu kullanılmakta ve böylece kod çözücü kısım Huffman kod kelimelerinin elde edilmesi işleminin bittiğini anlatmaktadır.

Bu doktora çalışmasında önerilen sistemi oluşturmak amacıyla araştırma ve inceleme aşamasında, öncelikle Türkçe metinlerdeki harf ve sembollerin dağılımından yola çıkılmış, daha sonra metinleri Türk dili kurallarına göre değerlendirebilmek için biçimbilimsel incelemesi yapılmıştır. Biçimbilimsel inceleme kapsamında Türkçe kelimeler, kök ve hecelerine ayrılmış ve tek başına kök olabilen harfler dışında köklerin ve hecelerin en az iki harf olmak üzere ortalama 3 harften oluştuğu gözlenmiştir. Veri sıkıştırma konusunda oluşturulan yöntemlerin başarısının değerlendirilmesinde kullanılmak üzere standart olarak kabul edilen Calgary Corpus ve Catenbury Corpus örnek alınarak 14 adet Türkçe metinden oluşturduğumuz *test-kümesi* üzerinde denemeler yapılıp, eleme yöntemi ile statik hece, kök-ek ve ilkhece-ek şablonları oluşturulup bunları kullanan farklı yöntemler ile metin dosyalarının sıkıştırılması gerçekleştirilmiştir.

İlk olarak Türkçe'deki karakterlerin kullanım sıklıklarından yararlanarak 107 elemanlı karaktere dayalı statik bir şablon oluşturulmuş ve *Karaktere Dayalı Statik Huffman* adını verdiğimiz yöntem ile *test-kümesi* içerisindeki metin dosyaları sıkıştırıldığında ortalama %45'lik bir sıkıştırma verimi elde edilmiştir. Aynı dosyalar *Karaktere Dayalı Dinamik Huffman* yöntemi ile sıkıştırıldığında sıkıştırma veriminde elde edilen başarı *Karaktere Dayalı Statik Huffman* yöntemindeki başarıdan az da olsa daha iyi sonuç vermiştir. Bunun sebebi dinamik Huffman kodlamasında karakterlerin kullanım sıklıklarının metine bağlı olarak değişmesidir. Daha sonra karakter sıkıştırmadan sembol sıkıştırmaya geçilmiş ve Türkçe'de en fazla sıklıkla kullanılan hecenin yüzde birinden küçük olan heceler hariç diğer karakterlerinde eklenmesiyle 578 elemanlı bir hece şablonu oluşturularak *test-kümesi*

içerisindeki metin dosyaları sıkıştırılmıştır. *Heceye Dayalı Statik Huffman* adını verdiğimiz bu yöntem ile metin dosyaları sıkıştırıldığında, sıkıştırma veriminde ortalama %47'lik bir başarı elde edilmiştir. Beklendiği gibi bu yöntem, hedeflenen biçimbilimsel sıkıştırma yöntemi geliştirilmesi kapsamında, karaktere dayalı sıkıştırmaya göre %2 daha iyi sonuç vermiştir. Sembol sıkıştırmada ikinci yöntem olarak kelimelerin kök ve eklerine ayrılarak kodlanması önerilmiş, Türkçe'deki kök ve eklerin çıkrılmasında (Eyüpoğlu Z., 1995), (Çotuksöken Y., 1991) ve (Hatiboğlu V., 1974) kaynaklarından yararlanılarak 478 elemanlı kök ve eklerden oluşan statik bir şablon oluşturularak *Köke Dayalı Statik Huffman* adını verdiğimiz bir yöntem ile *test-kümesi* içerisindeki metin dosyaları sıkıştırıldığında sıkıştırma veriminde ortalama %52'lik bir başarı elde edilmiştir. Bu aşamada araştırma bir adım daha ileri götürülerek, Türk Dil Kurumu imla kılavuzu ve sözlüğü temel alınmış ve Türk dilinde 5032 adet farklı hece olduğu ve bunlardan 2019 adedinin *test-kümesi* içerisindeki dosyalarda ilk hece olarak kullanıldığı belirlenmiştir. Bu sonuçdan yola çıkarak belirlenen 2019 adet hece *test-kümesi* içerisinde yer alan metinlerde taranmış ve kullanım sıklığı en fazla olan hecenin yüzde birinden küçük olanlar hariç, ek ve sembollerin de eklenmesiyle 533 elemanlı yeni bir şablon oluşturulup bu şablona göre sıkıştırma yapılmıştır. *İlk Heceye Dayalı Statik Huffman* adını verdiğimiz bu yöntem ile sıkıştırma veriminde ortalama %53'lük başarı elde edilmiştir.

Bu ön çalışmadan sonra önerilen kelimeye dayalı, kayıpsız dinamik sıkıştırma yöntemini oluşturmak amacı ile Türkçenin bitişken (Ünlü M., 1993) bir dil olmasından da yararlanılarak, kelimeler kök ve heceler yerine, birbirlerini kapsama durumuna göre en uzunu seçilecek şekilde kök-gövde ve eklerine ayrılmışlardır. Gövde yapısal olarak isim veya fiil köklerine bir takım yapım ekleri getirilerek oluşturulduğundan tek başına köklerden daha uzun olmakta ve böylece bit başına kodlanan bilgi miktarı artmaktadır. Kök-gövde'lerin yanı sıra ekler, karakterler, özel işaretler ile kodlama ve kod çözme aşamasında özel durumları ayırt etmek için kullanılan kontrol işaretleri metin içerisindeki kullanım sıklıklarına göre bağıl bir dağılım yapacak şekilde iki ayrı dinamik Huffman ağacına yerleştirilmişlerdir. Biçimbilimsel analizi yapıp, Türkçe bir kelimeyi kök-gövde ve eklerine ayırmak için Türk Dil Kurumu imla kılavuzundan yararlanarak 13.206 kelime ve 522 adeti ek olmak üzere 580 adet sembol içeren iki sözlük oluşturulmuştur. Bu sözlükler sayesinde sıkıştırılacak metin için kök ve gövdeleri içeren "G", sembolleri içeren "E" ağaçları dinamik Huffman

yöntemiyle oluşturulmuştur. Çözüm aşamasında, ağaçları yeniden oluşturmak amacıyla başlık kısmına, ağaçlardaki elemanların sıklık bilgisi konulması yoluna gidilmiş, fakat yerden kazanmak amacıyla sıklık bilgileri daha az bit ile ifade edilecek şekilde normalize edildiklerinde, sıkıştırma veriminin yapay olarak düştüğü gözlenince sıklık bilgisini göndermek yerine, (n) eleman için en fazla (2n-2) bit ile ifade edilebilecek, yöne göre çalışan Huffman ağacı kodlama yöntemi geliştirilmiştir. Bu yöntem aynı zamanda kod çözme aşamasında elemanları ağaca yerleştirmeden Huffman kod kelimelerinin de elde edilmesini sağlaması açısından hem yerden hem de zamandan tasarruf sağlamaktadır. Sıkıştırılmış bilginin önünde yer alan başlık bilgisi içerisinde, gönderilen sıklık bilgilerinin kapladığı alanın sadece %18 kullanılarak sıkıştırma veriminde ortalama %1,5 artış elde edilmiştir. Genişletme süresi de sıkıştırma süresinin yarısına indirilmiştir. *Geliştirilmiş Kelimeye Dayalı Dinamik Huffman* adını verdiğimiz bu yöntem 14 metinden oluşan *test-kümesi*'ne uygulandığında sıkıştırma verimi ortalama %59'a yükselmiştir. Geliştirilen bu yöntemle sıkıştırma veriminin (η) %50 ile %65 arasında değiştiği gözlenmiştir.

Geliştirilmiş Kelimeye Dayalı Dinamik Huffman yöntemi, biçimbilimsel yapıyı dikkate almadan sözlüğe dayalı sıkıştırma yapan yöntemlerden LZW'nin aynı *test-kümesi*'ndeki metinleri sıkıştırma verimi ile karşılaştırıldığında, ortalama %4 daha iyi sıkıştırma verimi elde etmiştir. Sıkıştırılan veriyi ikili kabul edip, dilbilgisi yapısına bağlı kalmadan sıkıştırma yapan yöntemlere göre sıkıştırma verimi ortalama %8 düşük çıkmakla beraber, karaktere dayalı yöntemlere göre sıkıştırma veriminde %13 daha fazla verim elde edilmiştir.

Sonuç olarak, bu tez çalışmasında Türkçe metinler biçimbilimsel yapılarına göre daha önce yapılan çalışmalarda örneği görülmeyen ve bu tez çalışmasında geliştirilen yöntem ile sıkıştırılmış ve çözülmüşlerdir. Buna göre;

1. Metinler Türk diline uygun şekilde kök-gövde ve eklerine ayrılmışlardır.
2. Elde edilen kök-gövde ve ekler metin içerisindeki kullanım sıklıklarına göre iki adet dinamik Huffman ağacına yerleştirilmişlerdir.
3. Çözüm esnasında ihtiyaç duyulan Huffman ağaç yapısı içerisindeki sıklık bilgisinin aktarılması maliyeti (n) eleman için (2n-2) biti geçmeyen ve doktora çalışmasında oluşturulan yöntemle gerçekleştirilmiştir.

Kök-gövde ve ek analizinde yararlanılan sözlüklerin gerektiğinde konuya özel metinlere uyum sağlayacak şekilde değiştirilebilir olması ve buna bağlı olarak sistem başarımının ölçeklenebilmesi de geliştirilen sistemin özgün özelliklerindendir.

KAYNAKLAR

Akın, H. L., Kuru, S., Güngör, T., Hamzaoğlu, İ. ve Arbatlı, D., (1993), "A Spelling Checker and Corrector for Turkish", Proceeding of the Second Turkish Symposium on Artificial Neural Networks, 24-25 June1993, Boğaziçi University, İstanbul, 113-119

Allen, J., (1987), Natural Language Understanding, The Benjamin/Cummings Publishing Company, Inc., California

Brodnik, A. ve Carlsson, S., (1998), "Sub-Linear Decoding of Huffman Codes Almost In-Place", Luleå University, Sweden

Burrows, M. ve Wheeler D., (1983) "Data Compression with the Burrows-Wheeler Transform", wysiwyg://46/http://www.dogma.net/markn/articles/bwt/bwt.htm

Compression Pointers, http://www.internz.com/compression-pointers.html

Compression Ratios, http://www.cs.waikato.ac.nz/~singlis/ratios.html

Cormen, T. H.; Leiserson, C. E. ve Rivest, R. L., (1990), Algorithms, Mc.Graw-Hill, NewYork

Çotuksöken, Y., (1991), Türkçede Ekler-Kökler-Gövdeler, Cem Yayınevi, İstanbul

Çölkesen, R., (1995), Sözlük Kullanan Sıkıştırma Algoritmaları için Yazılım ve Donanım Önerileri; V.42 bis Sıkıştırma Algoritmasını Gerçek Zamanda İşleyebilen Donanım, Ph.D. Tezi, Çukurova University, Adana

Data Compression, http://www.ics.uci.edu/~dan/pubs/DC-references.html

Demir, Ç. ve Oflazer, K., (1993), "An ATN Grammar for a Subset of Turkish", Proceeding of the Second Turkish Symposium on Artificial Neural Networks, 324-25 June 1993, Boğaziçi University, İstanbul, 162-169

Ergin, M., (1983), Türk Dil Bilgisi, Boğaziçi Yayınları, İstanbul

Eyüpoğlu, İ. Z, (1970), Türkçe Kökler Sözlüğü, Remzi Kitapevi, İstanbul

Fraenkel, A. S., Mor, M. ve Perl, Y., (1983), "Is Text Compression by Prefixes and Suffixes Pratical ?", Acta Information, 20:371-389

Graham, R. L., Knuth D. E. ve Patashnik, O., (1994) Concrete Mathematic (A Foundation for Computer Science), Addision Wesley, USA

Gottlieb, D., (1975), "A Classification of compression Methods and their Usefulness for a Large Data Processing Center", Procedings of National Computer Conference 44:453-458

Gonzales, C. R. ve Wintz, P., (1987), Digital Image Processing, Addison-Wesley, Canada

Gönenç, G., (1991), "A Finite-state Automation for Ssyllabification of Turkish words", Proceedings of the 1991 International Symposium on Computer and Information Sciences, November 1991, Antalya, 1039-1046

Gökçay, D. ve Halıcı, U., (1993), "A Morphological analysis of Turkish Words Using Neural Networks", Proceeding of the Second Turkish Symposium on Artificial Neural Networks, 24-25 June, 1993, Boğaziçi University, İstanbul, 129-133

Güngör, T., (1995), Computer Processing of Turkish:Morphological and Lexical Investigation, Ph.D. Thesis, Boğaziçi University, İstanbul

Güngör, T. ve Kuru, S., (1993), "Represantion of Turkish Morpholoy in ATN", Proceeding of the Second Turkish Symposium on Artificial Neural Networks, 24-25 June 1993, Boğaziçi University, İstanbul

Güngördü, Z. ve Oflazer, K., (1993), " A Lexical-Functional Grammer for a Subset of Turkish", Proceeding of the Second Turkish Symposium on Artificial Neural Networks, 24-25 June, 1993, Boğaziçi University, İstanbul

Hakkani, D. Z., Çiçekli, İ. ve Oflazer, K., (1996), Design and Implementation of a Tactical Generator for Turkish, a free Constituent Order Language, M.Sc. Thesis, Bilkent University, Ankara

Hamzaoğlu, İ. ve Kuru, S., (1993), "Machine Translation from Turkish to Its Dialects", Proceeding of the Second Turkish Symposium on Artificial Neural Networks, 24-25 June, Boğaziçi University, İstanbul, 135-145

Hsu, W. H., ve Zwarico, A. E., (1995), "Automatic Synthesis of Compression Techniques for Heterogeneous Files", Software-Practice and Experience, 25(10):1097-1116

Hatiboğlu, V., (1974), Türkçenin Ekleri, Ankara

Kibaroğlu, M. O., (1991), Spell Checking in Agglutinative Languages and Implementation for Turkish, M.S. Thesis, Boğaziçi University, İstanbul

Korkmaz, T. ve Çiçekli, İ., (1996), Turkish Text Generation with Systemic-Functional Grammar, M.Sc. Thesis, Bilkent University, Ankara

Kuruöz, İ. ve Oflazer, K. (1994), Tagging and Morphological Disambiguation of Turkish Text, M.Sc. Thesis, Bilkent University, Ankara

Knuth, D. E., (1985), "Dynamic Huffman Coding", Journal of Algorithms, 6:163-180

LZW Compression and Decompression,
http://www.tc.cornell/edu/Visualization/contrib/cs490-
94to95/yijou/myproj.html#dslzwalgo

Markley, R.W., (1990), Data Communications and Interoprerability, Prentice-Hall, London

Mcintyre, D. R. ve Pechure, M. A., (1985), "Data Compression Using Static Huffman Code-Decode Tables", Communication of ACM, 28:612-616

Salomon, D., (1998), Data Compression, Springer-Verlag Inc., New york

Sedgewick, R., (1988), Algorithms, Addision Wesley, New Jersey

Shooman, M. L., (1987), Software Engineering, Mc Graw Hill, NewYork

Tanaka, H., (1987), "Data Structure of Huffman Codes and Its Application to Efficient Encoding and Decoding", IEEE Transactions on Information Theory, 33:154-156

Türk Dil Kurumu, (1992), Türkçe Sözlük, Milliyet Tesisleri, İstanbul

Türk Dil Kurumu, (1996), İmla Kılavuzu, Türk Tarih Kurumu Basımevi, Ankara

Tür, G. ve Oflazer, K., (1996), Using Multiple Source of Information for Constraint-Based Morphological Disambiguation, M.Sc. Thesis, Bilkent University, Ankara

Ünlü, M., (1993), Dil Bilgileri, Cem Yayınevi, İstanbul

Wentian, L.,(1992), "Random Texts Exhibit Zipf's-Law-Like Word Frequency Distribution", IEEE Transactions on Information Theory IT-38(6),1842-1845

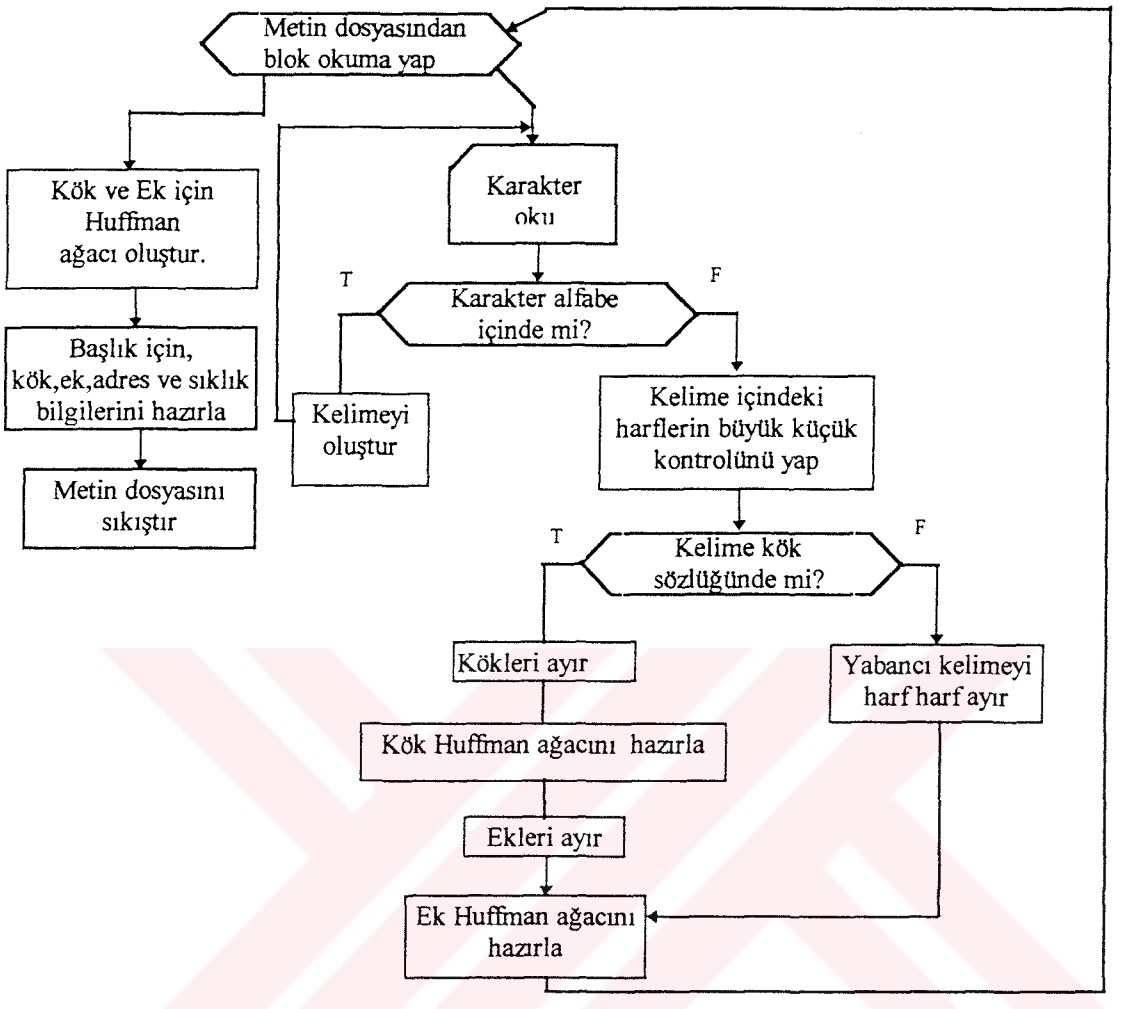
Witten, I. H, Neal, R. M. ve Cleary, J. G., (1987), "Arithmetic Coding for Data Compression", Communications of ACM, 30:520-539

Yorulmaz, K. A. ve Oflazer, K., (1997), Design and Implementation of a Computational Lexicon for Turkish, M.Sc. Thesis, Bilkent University, Ankara

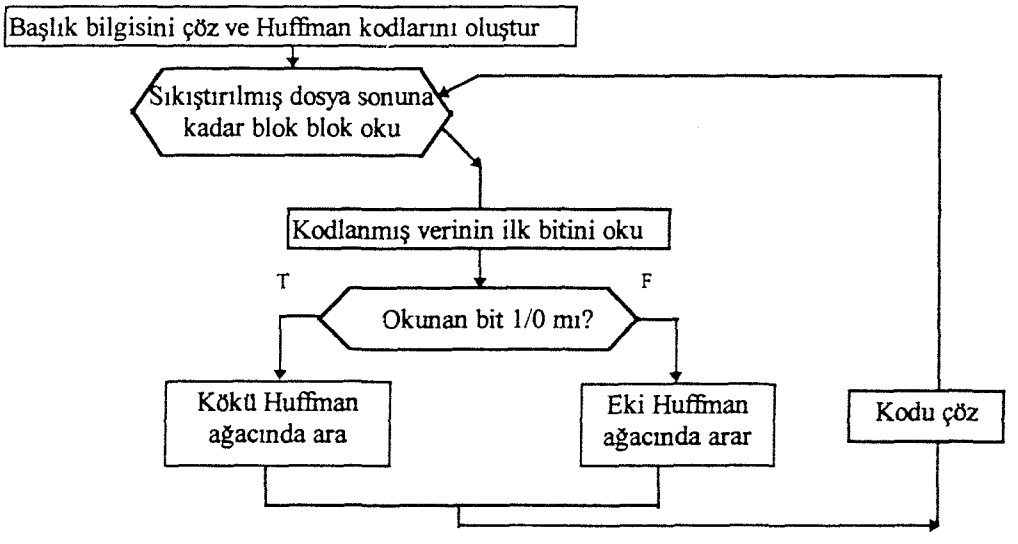
Ziv, J. ve Lempel, A., (1977), "A Universal Algoritm for Sequential Data Compression", IEEE Transaction on Information Theory IT-23(3), 337-343

EKLER

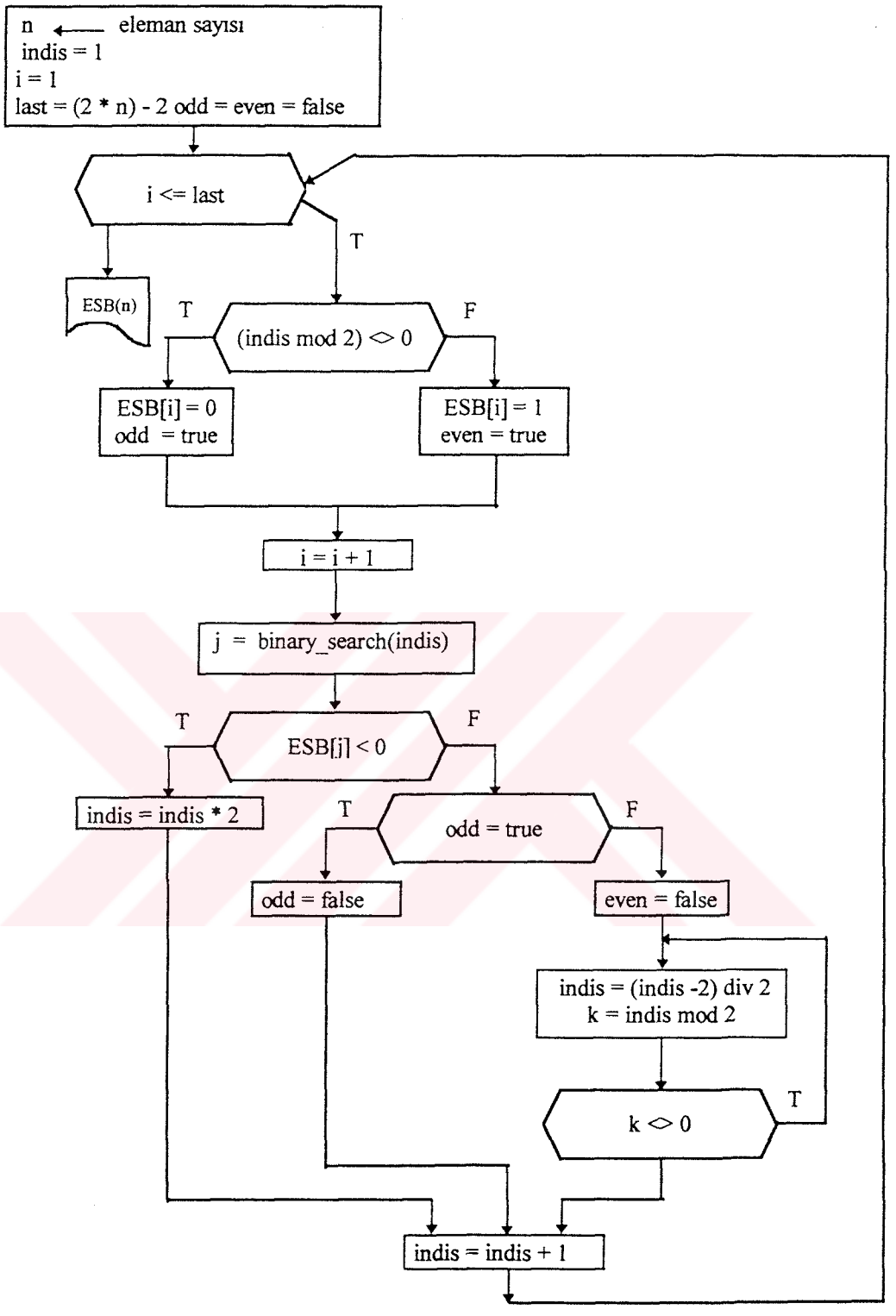
EK 1 GKeDDH Yönteminin Kodlama Bölümünün Blok Diyagramı



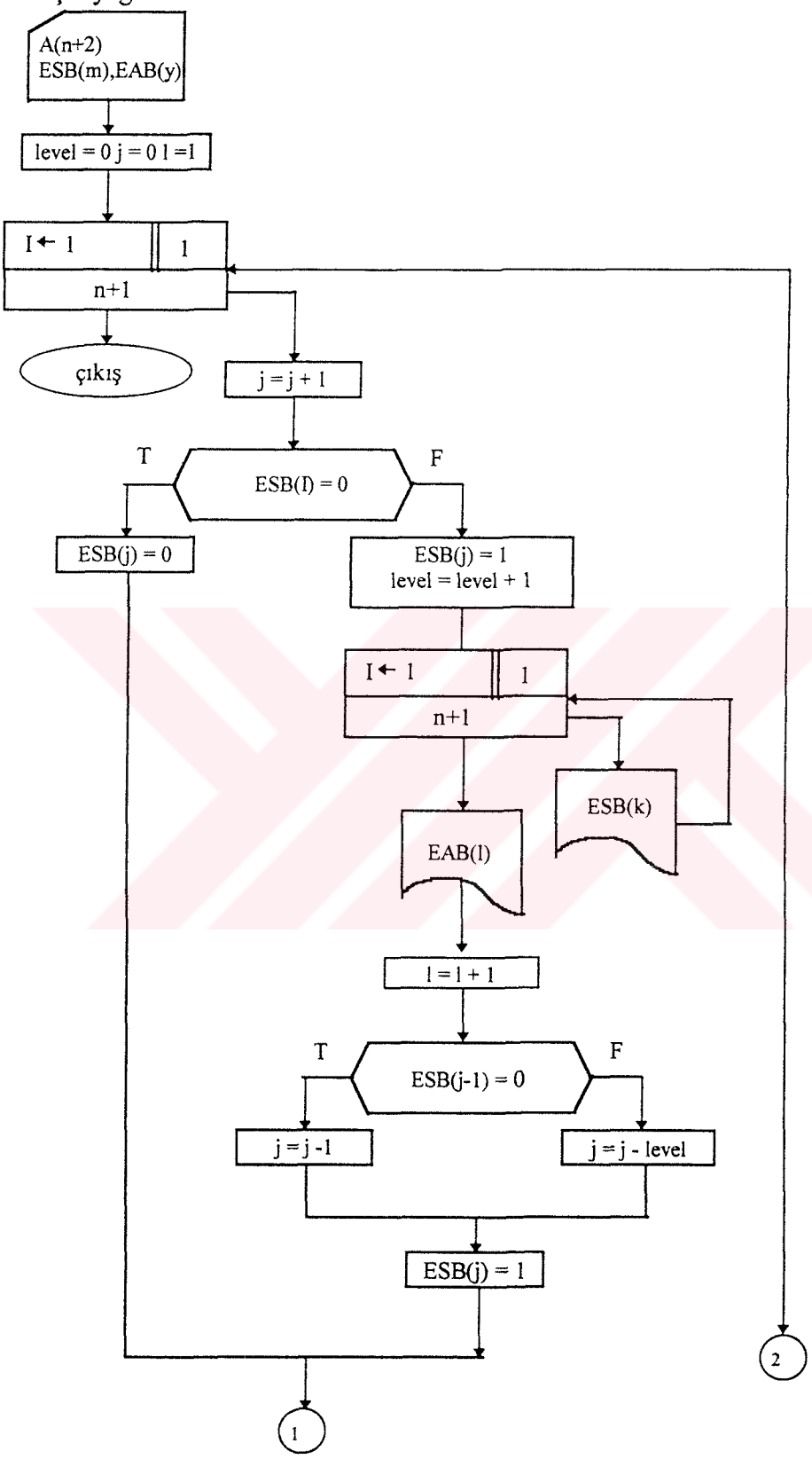
GKeDDH Yönteminin Kod Çözme Bölümünün Blok Diyagramı

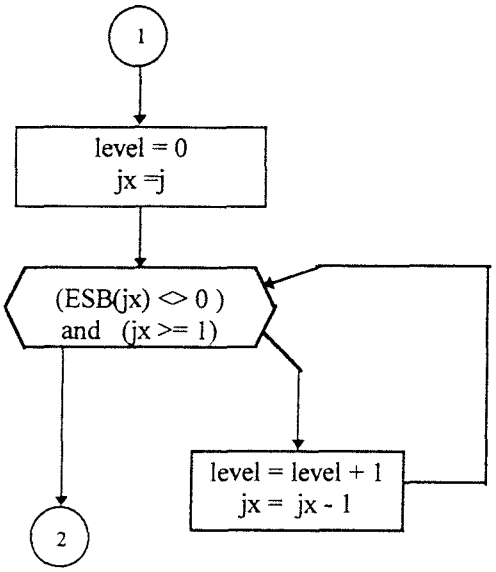


EK 2 Başlık'a gönderilecek olan sıklık bilgisinin oluşturulması için gerekli akış diyagramı



EK 3 Başlık'a gönderilen sıklık bilgisinin çözülüp, Huffman kodlarının elde edilmesi için gerekli akış diyagramı




1
level = 0
jx =j
(ESB(jx) <> 0)
and (jx >= 1)
level = level + 1
jx = jx - 1
2

EK 4 KrDSH Yönteminde Kullanılan Statik Şablon

Karakter	Sıklık	Olasılık	Karakter	Sıklık	Olasılık
#32	978853	0.292438	-	2541	0.000759
a	242724	0.072515	P	2493	0.000745
e	184937	0.055251	*	2279	0.000681
i	177194	0.052938	!	2117	0.000632
n	152265	0.04549	:	2075	0.00062
r	147641	0.044109	I	1827	0.000546
l	130801	0.039078	R	1673	0.000500
ı	107678	0.032169	Ş	1647	0.000492
k	92932	0.027764	9	1586	0.000474
d	88937	0.026570	Ç	1969	0.000588
m	74229	0.022176	Ö	1549	0.000463
y	72678	0.021713	C	1418	0.000424
u	69422	0.020740	0	1333	0.000398
t	67907	0.020288	V	1059	0.000316
#10	63617	0.019006	L	1029	0.000307
#13	63617	0.019006	2	1001	0.000299
s	59096	0.017655	F	933	0.000279
o	51352	0.015342	5	776	0.000232
b	47364	0.014150	8	701	0.000209
ü	41594	0.012426	j	700	0.000209
.	39493	0.011799	U	664	0.000198
ş	35029	0.010465	Ü	605	0.000181
z	32636	0.009750	Z	592	0.000177
,	25401	0.007589	I	575	0.000172
g	25030	0.007478	3	563	0.000168
'	24103	0.007201	7	551	0.000165
ç	21878	0.006536	)	482	0.000144
ğ	20736	0.006195	4	478	0.000143
h	20207	0.006037	(	482	0.000144
v	19908	0.005948	6	411	0.000123
c	18802	0.005617	;	153	4.66E-05
ö	16942	0.005062	J	122	3.64E-05
p	15759	0.004708	w	77	2.3E-05
B	10517	0.003142	Ğ	51	1.52E-05
A	8610	0.002572	/	34	1.02E-05
f	7786	0.002326	W	41	1.22E-05
K	5358	0.001601	q	11	3.29E-06
S	4771	0.001425	[	8	2.39E-06
T	4523	0.001351	]	8	2.39E-06
M	4289	0.001281	=	5	1.49E-06
İ	4045	0.001208	x	6	1.79E-06
H	3823	0.001142	Q	3	8.96E-07
E	3682	0.001100	X	3	8.96E-07
D	3675	0.001098	+	1	2.99E-07
Y	3649	0.001090		1	2.99E-07
"	3401	0.001016	@	1	2.99E-07
?	3303	0.000987	\	1	2.99E-07
O	3021	0.000903	{	1	2.99E-07
G	2770	0.000828	}	1	2.99E-07
N	2580	0.000771	&	1	2.99E-07

Karakter	Sıklık	Olasılık	Karakter	Sıklık	Olasılık
%	1	2.99E-07	é	1	2.99E-07
$	1	2.99E-07	<	1	2.99E-07
#	1	2.99E-07	>	1	2.99E-07
^	1	2.99E-07			

EK 5 HeDSH Yönteminde Kullanılan Statik Şablon

Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık
#32	978853	0.492604	ğı	4658	0.002344	çı	2138	0.001076
#13	63617	0.032015	gi	4655	0.002343	ci	2133	0.001073
la	23991	0.012073	mi	4652	0.002341	lik	2125	0.001069
le	21896	0.011019	nu	4624	0.002327	ğu	2087	0.001050
de	19676	0.009902	se	4617	0.002323	mu	2071	0.001042
ya	16748	0.008428	dan	4568	0.002299	mek	2071	0.001042
da	16653	0.008381	ği	4374	0.002201	al	2047	0.001030
a	14284	0.007188	nin	4108	0.002067	sın	2032	0.001023
ma	14097	0.007094	ko	3987	0.002006	kar	2027	0.001020
ka	13722	0.006906	ol	3787	0.001906	mak	2013	0.001013
di	13437	0.006762	lan	3751	0.001888	nun	2005	0.001009
bir	12066	0.006072	nın	3707	0.001866	tu	1947	0.000980
ri	11793	0.005935	ca	3696	0.001860	man	1939	0.000976
ra	11675	0.005875	baş	3580	0.001802	lık	1939	0.000976
i	10867	0.005469	ke	3549	0.001786	fa	1932	0.000972
ne	10618	0.005343	ö	3433	0.001728	rak	1916	0.000964
ki	10232	0.005149	ru	3179	0.001600	len	1916	0.000964
li	9583	0.004823	lu	3108	0.001564	va	1913	0.000963
bu	9517	0.004789	rın	3076	0.001548	şi	1909	0.000961
ye	9504	0.004783	gö	3055	0.001537	'	1905	0.000959
IB	9367	0.004714	ken	3025	0.001522	son	1889	0.000951
rı	9335	0.004698	,	3007	0.001513	var	1868	0.000940
me	9248	0.004654	ze	2998	0.001509	gün	1859	0.000936
si	9075	0.004567	za	2987	0.001503	bü	1839	0.000925
yor	8613	0.004334	mış	2984	0.001502	zı	1837	0.000924
o	8549	0.004302	ku	2979	0.001499	gü	1828	0.00092
sa	8298	0.004176	in	2977	0.001498	yıl	1825	0.000918
ve	8234	0.004144	tür	2931	0.001475	ay	1809	0.00091
na	8231	0.004142	mı	2926	0.001472	yan	1805	0.000908
te	8192	0.004123	miş	2922	0.001470	et	1782	0.000897
ni	8022	0.004037	şı	2864	0.001441	ça	1730	0.000871
ti	8021	0.004037	yi	2833	0.001426	çe	1678	0.000844
lar	7969	0.004010	be	2832	0.001425	lü	1654	0.000832
ta	7807	0.003929	dü	2757	0.001387	par	1648	0.000829
ba	7771	0.003911	kı	2742	0.001380	ver	1636	0.000823
bi	7691	0.003870	rin	2694	0.001356	rum	1627	0.000819
nı	7663	0.003856	yo	2633	0.001325	çık	1618	0.000814
re	7236	0.003641	yı	2608	0.001312	dir	1596	0.000803
dı	6875	0.003460	u	2590	0.001303	mü	1565	0.000788
ge	6581	0.003312	so	2536	0.001276	yu	1556	0.000783
sı	6458	0.003250	pa	2519	0.001268	do	1553	0.000782
ha	6330	0.003186	çin	2502	0.001259	her	1545	0.000778
lı	6301	0.003171	is	2471	0.001244	sin	1544	0.000777
ler	6277	0.003159	cak	2425	0.001220	let	1542	0.000776
e	5806	0.002922	an	2418	0.001217	dar	1530	0.000770
du	5482	0.002759	kü	2320	0.001168	cı	1523	0.000766
ce	5337	0.002686	tan	2303	0.001159	dır	1518	0.000764
tı	5296	0.002665	kan	2292	0.001153	yap	1511	0.000760
.	5080	0.002556	su	2278	0.001146	çi	1508	0.000759
den	4894	0.002463	şa	2255	0.001135	maz	1507	0.000758

Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık
san	1498	0.000754	lur	631	0.000318	gu	618	0.000311
zi	1488	0.000749	el	625	0.000315	mar	477	0.000240
“	1483	0.000746	lış	622	0.000313	mok	475	0.000239
kal	1453	0.000731	lim	612	0.000308	iz	470	0.000237
tü	1441	0.000725	gı	606	0.000305	müş	461	0.000232
yet	1429	0.000719	sıl	603	0.000303	liş	461	0.000232
nız	954	0.000480	ğın	603	0.000303	hem	461	0.000232
sun	942	0.000474	oy	600	0.000302	mer	460	0.000231
biz	938	0.000472	ğü	586	0.000295	fe	460	0.000231
hu	931	0.000469	zü	585	0.000294	mec	459	0.000231
dün	916	0.000461	tik	584	0.000294	hep	454	0.000228
ruz	909	0.000457	bay	583	0.000293	şün	452	0.000227
miz	889	0.000447	as	582	0.000293	nen	452	0.000227
ak	887	0.000446	kul	579	0.000291	har	451	0.000227
at	880	0.000443	sim	575	0.000289	bek	450	0.000226
mız	853	0.000429	tın	569	0.000286	ilk	449	0.000226
top	847	0.000426	öğ	566	0.000285	dim	444	0.000223
ül	842	0.000424	bin	565	0.000284	mal	443	0.000223
zel	839	0.000422	dım	564	0.000284	dil	443	0.000223
rar	839	0.000422	üs	558	0.000281	kö	442	0.000222
nan	836	0.000421	zim	553	0.000278	sağ	440	0.000221
dö	836	0.000421	luk	553	0.000278	kay	432	0.000217
bul	835	0.00042	kat	553	0.000278	ev	432	0.000217
şey	833	0.000419	to	552	0.000278	tim	430	0.000216
nel	804	0.000405	zan	549	0.000276	duy	430	0.000216
lay	804	0.000405	uy	549	0.000276	nim	429	0.000216
tık	795	0.0004	tar	549	0.000276	ber	429	0.000216
hal	703	0.000354	nün	547	0.000275	lam	427	0.000215
lın	701	0.000353	git	546	0.000275	kil	427	0.000215
bo	693	0.000349	lo	540	0.000272	tıl	424	0.000213
yen	692	0.000348	geç	538	0.000271	vu	421	0.000212
lin	691	0.000348	lun	536	0.00027	lem	421	0.000212
şan	683	0.000344	han	534	0.000269	hat	421	0.000212
ğer	681	0.000343	şın	529	0.000266	lır	417	0.000210
ret	673	0.000339	ah	527	0.000265	ro	415	0.000209
dık	667	0.000336	lah	519	0.000261	rat	413	0.000208
ek	665	0.000335	kin	519	0.000261	çil	413	0.000208
dur	664	0.000334	leş	512	0.000258	öz	408	0.000205
yol	660	0.000332	gös	510	0.000257	çö	408	0.000205
fi	660	0.000332	lım	508	0.000256	rıl	407	0.000205
hi	659	0.000332	run	504	0.000254	ı	407	0.000205
gör	659	0.000332	fin	503	0.000253	ik	406	0.000204
dın	659	0.000332	sal	500	0.000252	yaz	404	0.000203
cum	650	0.000327	pe	499	0.000251	tak	402	0.000202
nem	648	0.000326	şü	497	0.00025	öl	401	0.000202
dam	647	0.000326	vet	491	0.000247	daş	400	0.000201
yon	645	0.000325	rul	491	0.000247	gin	397	0.000200
mah	643	0.000324	say	490	0.000247	ber	429	0.000216
ğa	639	0.000322	öy	486	0.000245	lam	427	0.000215
ço	636	0.000320	ok	482	0.000243	kil	427	0.000215
zin	635	0.000320	çim	482	0.000243	tıl	424	0.000213
laş	634	0.000319	ril	481	0.000242	vu	421	0.000212

Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık
lem	421	0.000212	duk	347	0.000175	av	284	0.000143
hat	421	0.000212	lük	343	0.000173	raz	283	0.000142
öz	408	0.000205	suz	339	0.000171	rir	282	0.000142
çö	408	0.000205	şin	337	0.00017	sut	281	0.000141
rıl	407	0.000205	ral	333	0.000168	un	280	0.000141
ı	407	0.000205	ğır	333	0.000168	ses	279	0.00014
ik	406	0.000204	nım	332	0.000167	lak	278	0.00014
yaz	404	0.000203	bak	331	0.000167	raf	277	0.000139
tak	402	0.000202	ğe	329	0.000166	hay	277	0.000139
öl	401	0.000202	şam	327	0.000165	nap	276	0.000139
daş	400	0.000201	ker	327	0.000165	?	276	0.000139
gin	397	0.0002	üç	326	0.000164	riy	275	0.000138
göz	393	0.000198	id	324	0.000163	vur	273	0.000137
gül	392	0.000197	kıl	323	0.000163	sis	273	0.000137
zar	391	0.000197	nır	322	0.000162	kış	272	0.000137
kir	391	0.000197	-	322	0.000162	zor	270	0.000136
kaç	391	0.000197	sıy	320	0.000161	van	270	0.000136
ih	391	0.000197	hur	320	0.000161	şık	270	0.000136
nuş	390	0.000196	düş	320	0.000161	:	270	0.000136
mut	390	0.000196	rim	316	0.000159	des	269	0.000135
mo	387	0.000195	ket	316	0.000159	muz	268	0.000135
nuz	386	0.000194	kır	315	0.000159	ğun	266	0.000134
ağ	386	0.000194	ir	315	0.000159	can	266	0.000134
bağ	385	0.000194	net	311	0.000157	kut	265	0.000133
tam	383	0.000193	gir	311	0.000157	rıy	264	0.000133
ğım	383	0.000193	yıp	309	0.000156	nıt	263	0.000132
zen	382	0.000192	bey	307	0.000154	yip	262	0.000132
yal	381	0.000192	ner	306	0.000154	em	262	0.000132
sen	381	0.000192	im	305	0.000153	ven	261	0.000131
rım	381	0.000192	gun	302	0.000152	aç	261	0.000131
es	381	0.000192	*	302	0.000152	kes	260	0.000131
ser	379	0.000191	suç	301	0.000151	hür	260	0.000131
lat	379	0.000191	nok	301	0.000151	ab	260	0.000131
tem	376	0.000189	tal	296	0.000149	vaş	258	0.00013
set	374	0.000188	kor	296	0.000149	muz	268	0.000135
sev	372	0.000187	fi	296	0.000149	ğun	266	0.000134
kez	371	0.000187	pi	294	0.000148	can	266	0.000134
dür	371	0.000187	sel	293	0.000147	kut	265	0.000133
rev	370	0.000186	sak	293	0.000147	rıy	264	0.000133
cü	369	0.000186	bet	293	0.000147	nıt	263	0.000132
yin	364	0.000183	sö	292	0.000147	yip	262	0.000132
rüş	364	0.000183	res	292	0.000147	em	262	0.000132
pan	363	0.000183	rel	291	0.000146	ven	261	0.000131
rün	356	0.000179	HB	291	0.000146	aç	261	0.000131
yun	353	0.000178	no	288	0.000145	kes	260	0.000131
yak	353	0.000178	çen	288	0.000145	hür	260	0.000131
fah	353	0.000178	cuk	288	0.000145	ab	260	0.000131
pıl	352	0.000177	pek	287	0.000144	vaş	258	0.00013
zın	351	0.000177	tüm	285	0.000143	nut	255	0.000128
gön	350	0.000176	sek	285	0.000143	pat	253	0.000127
mın	349	0.000176	kam	285	0.000143	kav	251	0.000126
zın	347	0.000175	zün	284	0.000143	dip	251	0.000126

Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık
çü	250	0.000126	7	42	2.11E-05	q	31	1.56E-05
sor	248	0.000125	b	31	1.56E-05	w	31	1.56E-05
nek	246	0.000124	c	31	1.56E-05	x	31	1.56E-05
luş	245	0.000123	ç	31	1.56E-05	+	31	1.56E-05
bah	244	0.000123	d	31	1.56E-05	/	31	1.56E-05
gür	243	0.000122	f	31	1.56E-05	#	31	1.56E-05
fet	242	0.000122	g	31	1.56E-05	$	31	1.56E-05
çar	242	0.000122	ğ	31	1.56E-05	&	31	1.56E-05
bas	242	0.000122	h	31	1.56E-05	%	31	1.56E-05
bö	240	0.000121	j	31	1.56E-05	\	31	1.56E-05
1	171	8.61E-05	k	31	1.56E-05	{	31	1.56E-05
0	163	8.2E-05	l	31	1.56E-05	}	31	1.56E-05
!	135	6.79E-05	m	31	1.56E-05		31	1.56E-05
2	86	4.33E-05	n	31	1.56E-05	<	31	1.56E-05
5	76	3.82E-05	p	31	1.56E-05	>	31	1.56E-05
(	54	2.72E-05	r	31	1.56E-05	@	31	1.56E-05
)	54	2.72E-05	s	31	1.56E-05	[	31	1.56E-05
;	49	2.47E-05	ş	31	1.56E-05	]	31	1.56E-05
4	49	2.47E-05	t	31	1.56E-05	=	31	1.56E-05
8	47	2.37E-05	v	31	1.56E-05	6	31	1.56E-05
3	46	2.31E-05	y	31	1.56E-05			
9	46	2.31E-05	z	31	1.56E-05			

EK 6 Kök Listesi

alk alt ard art ayr
ab ac aç ad af ag ağ ah ak al am an ap ar as aş at av ay az
a
balk berk burk
bac bad bağ bah bak bal bam ban bar bas baş bat bav bay baz
bec bed beg beğ bek bel ben ber bes beş bet bey bez
bıc bıç bık bıl bım bın bır bıs bış bıt biç
bil bin bir bis biş bit biz
bod bog boğ bok bol boş boy boz
böc böd böğ bör
buç bud buğ buk bul bum bun bur buş but buv buy buz
büd büg büğ bük bür büs büş büt büv büy büz
be
b
cab cav caz cıv cız coz cuz cüv cüy cüz
c
çarp çırp
çab çaç çad çaf çag çağ çak çal çan çap çar ças çaş çat çav çay çaz
çeb çeğ çek çel çem çen çep çer çes çeş çet çev çey çez
çıb çığ çık çıl çım çın çıp çır çıs çış çıt çıv çıy çız
çiğ çil çim çiş çit çiv çiy çiz
çoğ çok çol
çög çöğ çök çöm çön çör çöş çöt çöv çöy çöz
çub çuğ çuk çul çum çun çup çur çus çuş çut çuv çuy çuz
çük çül çüm çün çür çüs çüş çüv çüy
ça çe çü
damz depr devş doğr dürt
dad dağ dak dal dam dan dap dar dav day daz
deb ded değ deh dek del dem den dep der deş dev dey
dıb dığ dık dıl dım dın dır dış dız
dib did dig diğ dik dil dim din dip dir diş dit div diy diz
doğ dok dol dom don dop dor dos doy
döğ dök döl dön dör döş döv
dud dul dum dur duy
düğ düm dün dür düş düt düv düz
d
eğr eks emz ert esn esr
ed eg eğ ek el em en ep er es eş et ev ey ez
e
far
fır fıs fış fıt fıv fıy fız
fos
fös

fus
füs füv füy
f
gevr gevş
gönd göst
gag gak
geb gec geç ged geğ gel gen gep ger get gev gey gez
gıc gıd gıg gığ gıh gık gır gıs gış gıt gıv gıy gız
gid gir git giy giz
goc
göb göc göç göğ gök göl göm gön gör gös göt göv göy göz
gud gug guk gul guz
güc güç güd gül güm gün güp gür güt güv güz
g
h
ıb ıc ıd ığ ık ıl ım ıp ır ıs ış ıt ıv ıy ız
ı
iğr imr irt ist
ib iç iğ ik il im in ip ir is iş it iv iy iz
i
kalk kavr kavş kayn kayt
kert
kırk kırp kışk
kösn
kurt kuyr
kükr
kab kaç kad kağ kak kal kam kan kap kar kas kaş kat kav kay kaz
keb keç kek kem ken kep ker kes keş ket kev key kez
kıb kıc kıç kıd kığ kık kıl kım kın kıp kır kıs kış kıt kıv kıy kız
kiç kik kim kin kip kir kis kiş kit kiz
koc koç koğ kok kol kom kon kop kor kos koş kot kov koy koz
köç kök köl köm kön köp kör kös köt köv köy köz
kuc kud kuğ kuk kul kum kun kup kur kus kuş kut kuy kuz
küç küf kük kül küm kün küp kür küs küş küt küv küy küz
ka
k
l
m
n
ob oc od of oğ ok ol om on op or os ot ov oy oz
öğr ölç ört
öb öc öç öd öf öğ ök öl öm ön öp ör ös öş öt öv öy öz
ö
paf par pas pat pav
pef peh pek
pır pıs pış pıt

pin pis piş
poh por pot
pöf pör
püf pür püs püt
puf pus puş
pa
p
r
salt sark sars
serp seyr
silk
sürç sürt
sab sac saç sag sağ sak sal sam san sap sar sas sat sav say
seç seg seğ sek sel sem sen sep ser ses seş sev sey sez
sıb sıc sıç sıd sığ sık sım sın sıp sır sıt sıv sıy sız
sid sig siğ sik sil sim sin sip sis sit siv siy siz
sog soğ sok sol som son sop sor sos sov soy
söb söğ sök söl söm sön sör söv söy söz
sub suc suç sud suk sul sum sun sur sus suv suy
süc süç süd süğ sük sül süm sün süp sür süs süt süv süy süz
s
şab şak şar şaş şeş
şık şıl şıp şır
şim şiş
şor şur
ş
tart ters
tab tad tağ tak tal tam tan tap tar tas taş tat tav tay taz
tec ted teğ tek tel tem ten tep ter tes teş tet tev tey tez
tıd tıg tığ tık tıl tım tın tıp tır tıs tış tıt tıv tıy tız
tig tiğ tik til tim tin tir tis tiş tit tiv tiy tiz
tod toğ tok tol tom ton top tor tos toş tot tov toy toz
töb tög töğ tök töl töm tön töp tör tös töş töy töz
tub tug tuğ tuh tuk tul tum tun tup tur tus tuş tut tuv tuy tuz
tüb tüg tüğ tük tül tüm tün tüp tür tüs tüş tüt tüv tüy tüz
t
uğr
ub uc uç ud uf ug uğ uk ul um un up ur us uş ut uv uy uz
u
ürk üst
üb üc üç üd üf üğ ük ül üm ün üp ür üs üş üt üv üy üz
ü
vağ van var ver vıc vıd vığ vın vır vıy vız vur
v
yırt yüks
yab yac yad yaf yag yağ yak yal yam yan yap yar yas yaş yat yav yay yaz

yeb yed yeg yeğ yek yel yem yen yep yer yes yeş yet yev yey
yıd yıg yığ yık yıl yım yın yıp yır yış yıt yıv
yid yiğ yik yil yim yin yip yir yiş yit yiv yiy
yoc yod yog yoğ yok yol yom yon yop yor yos yoş yot yov yoy yoz
yög yöğ yök yöl yön yör
yub yud yuf yug yuğ yuk yul yum yun yup yur yus yuş yut yuv yuy
yüc yüd yüğ yük yül yüm yün yür yüş yüt yüv yüy yüz
zart zurt
zır zor

EK 7 Ek Listesi

acak acan adak agan ağan akla alak alga amaç amak amık anak arak
ağı ala alı ama arı ası
aç ak al am an ar aş at av ay az
a
baç beç buz büz
cılayın cileyin
casına cesine
cacık cağız cecik ceğiz cığaz ciğez cuğaz cüğez
cama cana casi ceme cene cesi
cak can cek cen cık cıl cım cın cik cil cim cin cuk cul cum cun cük cül cüm cün
ca ce cı ci cu cü
çacık çağız çecik çeğiz
çana çası çene çesi
çak çek çık çıl çım çın çik çil çim çin çuk çul çum çun çük çül çüm çün
ça çe çı çi çu çü
ç
dırlar dirler durlar dürler
dalık delik dıkça dıkta dırık dikçe dikte dirik dukça dukta duruk dürük dükçe dükte
daki deki
dak dam dan dar daş dat dek dem den der deş det dık dır dız dik dir diz duk dur duz dük
dür düz
da de dı di du dü
ecek ecen edek egen eğen ekle elek elge elim emeç emek emik enek erek
eği ele eli eme eri esi
eç ek el em en er eş et ev ey ez
e
gaç gan geç gen gıç gıl gın gıt giç gil gin git guç gul gun gur gut güç gül gün gür güt
ga ge gı gi gu gü
ımtırak
ımsa ınca ınız ırga ıyor
ılı
ıç ık ıl ım ın ıp ır ış ız
ıcı
ı
imtırak
imse ince iniz irge iyor
ici ili
iç ik il im in ip ir iş iz
i
ktir
kaç kan keç kek ken kıl kın kır kil kin kir kit kul kun kur kül kün kür
ka ke kı ki ku kü

k
layın leyin
lama ları leme leri
lak lam lan lar laş laz lek lem len ler leş lez lık lım lik lim luk lum lük
la le lı li lu lü
l
mazlık mezlik mtırak
madan makta meden mekte
maca mala malı mece mele meli msar mser
maç mak man mar maş maz meç mek men mer meş mez mık miş mız mik miş miz muk mur muş
muz mük mür müş müz msa mse msı msi msu msü
ma me mı mi mu mü
m
nak nek ncı nci ncu ncü nın nız nin niz ntı nti ntu ntü nun nuz nün nüz
nç
n
pak pek
p
rak rek rga rge
ra re
r
sınlar sinler sunlar sünler sınız sızın siniz sunuz sünüz
sak sal sek sel sık sıl sın sız sil sin siz sul sun suz sül sün süz
sa se sı si su sü
ştır ştir ştur ştür
şar şer şın şin
ş
tırlar tirler turlar türler tıkça tıkta tikçe tikte tukça tukta tükçe tükte
taki teki
tam tan tar taş tay tem ten ter teş tık tır tız tik tir tiz tuk tur tuz tük tür tüz
ta te tı ti tu tü
t
umsa unca unuz uyor
ucu ulu
uç uk ul um un up ur uş uz
u
ümse ünce ünüz üyor
ücü ülü
üç ük ül üm ün üp ür üş üz
ü
van ven
v
yor
y
zir
z

EK 8 KökDSH Yönteminde Kullanılan Statik Şablon

Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık
#32	978853	0.564822	den	3337	0.001926	s	1790	0.001033
#13	63617	0.036709	li	3230	0.001864	aç	1766	0.001019
a	20509	0.011834	ıl	3190	0.001841	ed	1760	0.001016
i	17293	0.009978	sa	3083	0.001779	ön	1759	0.001015
e	15552	0.008974	yor	3055	0.001763	on	1744	0.001006
n	14465	0.008347	lı	3055	0.001763	son	1737	0.001002
ı	14158	0.008170	m	3048	0.001759	em	1737	0.001002
y	13482	0.007779	el	3047	0.001758	değ	1719	0.000992
in	12700	0.007328	,	3007	0.001735	sı	1718	0.000991
an	11697	0.006749	ak	2979	0.001719	uz	1706	0.000984
bir	10605	0.006119	ra	2934	0.001693	se	1684	0.000972
ın	10424	0.006015	ka	2870	0.001656	acak	1684	0.000972
ğ	10374	0.005986	am	2837	0.001637	kon	1662	0.000959
IB	9367	0.005405	yap	2835	0.001636	var	1654	0.000954
di	7491	0.004322	iç	2766	0.001596	l	1646	0.000950
u	7293	0.004208	iyor	2694	0.001555	ası	1616	0.000932
ol	7086	0.004089	baş	2615	0.001509	z	1612	0.000930
ar	6960	0.004016	um	2559	0.001477	gün	1580	0.000912
r	6659	0.003842	si	2534	0.001462	kad	1564	0.000902
il	6571	0.003792	h	2500	0.001443	diy	1541	0.000889
da	6526	0.003766	ş	2488	0.001436	ger	1539	0.000888
lar	6211	0.003584	mış	2468	0.001424	aş	1508	0.000870
en	6086	0.003512	at	2449	0.001413	dır	1499	0.000865
dı	6031	0.003480	ü	2443	0.001410	"	1483	0.000856
ları	5446	0.003142	ken	2421	0.001397	te	1472	0.000849
ma	5431	0.003134	ış	2402	0.001386	yan	1469	0.000848
un	5373	0.003100	gör	2393	0.001381	or	1468	0.000847
b	5248	0.003028	ki	2380	0.001373	bel	1464	0.000845
.	5080	0.002931	ek	2371	0.001368	lan	1456	0.000840
t	4958	0.002861	ım	2356	0.001359	ta	1426	0.000823
o	4672	0.002696	ıyor	2318	0.001338	p	1409	0.000813
du	4509	0.002602	ver	2306	0.001331	dir	1409	0.000813
ay	4303	0.002483	kar	2279	0.001315	par	1369	0.000790
de	4279	0.002469	eri	2235	0.001290	kan	1329	0.000767
im	4219	0.002434	ul	2230	0.001287	uş	1318	0.000761
la	4214	0.002432	ik	2182	0.001259	dur	1311	0.000756
er	4151	0.002395	miş	2172	0.001253	çok	1311	0.000756
leri	4117	0.002376	ır	2116	0.001221	kal	1309	0.000755
ti	4012	0.002315	bil	2106	0.001215	ala	1297	0.000748
le	3986	0.002300	f	2096	0.001209	ük	1270	0.000733
ler	3947	0.002278	sın	2078	0.001199	tan	1270	0.000733
ün	3867	0.002231	gel	2069	0.001194	ili	1260	0.000727
et	3767	0.002174	arı	1946	0.001123	ca	1253	0.000723
tı	3754	0.002166	'	1905	0.001099	lu	1247	0.000720
al	3702	0.002136	iz	1893	0.001092	ist	1228	0.000709
dan	3581	0.002066	ık	1883	0.001087	ecek	1225	0.000707
me	3490	0.002014	sin	1878	0.001084	ad	1222	0.000705
k	3426	0.001977	çık	1867	0.001077	ev	1219	0.000703
iş	3412	0.001969	ce	1865	0.001076	ke	1205	0.000695
ir	3396	0.001960	ur	1812	0.001046	sor	1196	0.000690

Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık
düş	1184	0.000683	dik	780	0.00045	kul	566	0.000327
mak	1180	0.000681	lam	778	0.000449	üyor	563	0.000325
lık	1174	0.000677	büy	777	0.000448	üş	562	0.000324
yer	1172	0.000676	dön	776	0.000448	seç	556	0.000321
bun	1161	0.000670	is	775	0.000447	ağı	550	0.000317
arak	1159	0.000669	sür	774	0.000447	d	548	0.000316
geç	1143	0.000660	üm	768	0.000443	as	548	0.000316
ül	1136	0.000655	esi	759	0.000438	tur	544	0.000314
tar	1124	0.000649	der	759	0.000438	alt	542	0.000313
ama	1116	0.000644	su	757	0.000437	ele	538	0.000310
tır	1115	0.000643	yaş	756	0.000436	akla	530	0.000306
ey	1102	0.000636	yak	756	0.000436	ulu	529	0.000305
ür	1101	0.000635	lama	756	0.000436	tut	529	0.000305
yaz	1079	0.000623	siz	748	0.000432	bin	529	0.000305
maz	1070	0.000617	uy	739	0.000426	tem	526	0.000304
kur	1058	0.00061	top	739	0.000426	git	524	0.000302
lik	1047	0.000604	gil	739	0.000426	ter	516	0.000298
uyor	1046	0.000604	ded	723	0.000417	öl	516	0.000298
bak	1041	0.000601	eme	716	0.000413	deki	514	0.000297
muş	1012	0.000584	nın	713	0.000411	doğr	511	0.000295
sun	1007	0.000581	kü	702	0.000405	cı	510	0.000294
biz	988	0.000570	yol	694	0.00040	san	502	0.000290
tir	985	0.000568	tay	688	0.000397	iy	501	0.000289
alı	978	0.000564	ok	679	0.000392	dün	500	0.000289
bul	972	0.000561	re	670	0.000387	up	496	0.000286
az	958	0.000553	dol	666	0.000384	kes	495	0.000286
yok	955	0.000551	duy	662	0.000382	gi	491	0.000283
çek	954	0.000550	ö	645	0.000372	bağ	491	0.000283
tu	950	0.000548	ağ	645	0.000372	üst	485	0.000280
dem	943	0.000544	söz	644	0.000372	nız	483	0.000279
ıp	941	0.000543	göz	636	0.000367	leme	482	0.000278
ip	941	0.000543	tek	633	0.000365	tür	481	0.000278
mi	919	0.000530	nin	633	0.000365	ge	480	0.000277
dü	917	0.000529	kaç	631	0.000364	dür	478	0.000276
ız	914	0.000527	ci	629	0.000363	mez	477	0.000275
yüz	912	0.000526	gir	627	0.000362	yet	473	0.000273
len	909	0.000525	laş	625	0.000361	kor	466	0.000269
mek	894	0.000516	siy	624	0.000360	kat	463	0.000267
üz	873	0.000504	eş	614	0.000354	büt	463	0.000267
ılı	871	0.000503	tü	613	0.000354	kız	461	0.000266
söy	864	0.000499	gid	606	0.000350	düz	458	0.000264
yar	859	0.000496	say	605	0.000349	mı	454	0.000262
v	847	0.000489	ten	604	0.000349	kiş	454	0.000262
ben	847	0.000489	yen	603	0.000348	dık	449	0.000259
yıl	843	0.000486	eği	601	0.000347	ot	446	0.000257
lü	842	0.000486	daki	599	0.000346	sız	443	0.000256
ç	838	0.000484	çal	592	0.000342	sır	441	0.000254
man	835	0.000482	yön	588	0.000339	oy	439	0.000253
kim	822	0.000474	get	587	0.000339	din	436	0.000252
dev	811	0.000468	uk	573	0.000331	tam	432	0.000249
kap	801	0.000462	sal	571	0.000329	öz	431	0.000249
gen	793	0.000458	bas	568	0.000328	men	431	0.000249

Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık
eli	429	0.000248	kab	327	0.000189	ab	243	0.000140
erek	428	0.000247	yas	325	0.000188	küç	242	0.000140
göst	425	0.000245	ör	325	0.000188	lük	241	0.000139
baz	424	0.000245	sar	324	0.000187	far	240	0.000138
sav	423	0.000244	sak	324	0.000187	nun	239	0.000138
şim	420	0.000242	öt	323	0.000186	yem	238	0.000137
ga	414	0.000239	-	322	0.000186	malı	238	0.000137
es	410	0.000237	ici	317	0.000183	it	236	0.000136
sev	409	0.000236	kalk	315	0.000182	kez	235	0.000136
leş	408	0.000235	dar	314	0.000181	kas	235	0.000136
ban	405	0.000234	gönd	311	0.000179	ber	234	0.000135
uç	403	0.000233	ıcı	310	0.000179	dak	233	0.000134
cak	399	0.000230	güz	309	0.000178	niz	231	0.000133
boy	399	0.000230	miz	308	0.000178	day	230	0.000133
kil	398	0.000230	kaz	307	0.000177	kıl	229	0.000132
çe	398	0.000230	suz	305	0.000176	gür	229	0.000132
tak	395	0.000228	ser	304	0.000175	lak	228	0.000132
ses	392	0.000226	gul	302	0.000174	sü	227	0.000131
lem	392	0.000226	*	302	0.000174	ünce	226	0.000130
pa	389	0.000224	gun	301	0.000174	ülü	226	0.000130
daş	389	0.000224	ça	298	0.000172	kın	224	0.000129
öğr	388	0.000224	sil	297	0.000171	mele	215	0.000124
art	388	0.000224	öy	297	0.000171	ince	213	0.000123
müş	382	0.000220	kit	297	0.000171	rak	212	0.000122
yür	381	0.000220	madan	291	0.000168	unca	211	0.000122
taş	381	0.000220	HB	291	0.000168	tab	211	0.000122
tık	380	0.000219	tik	287	0.000166	makta	211	0.000122
sağ	380	0.000219	ac	284	0.000164	dil	207	0.000119
yay	378	0.000218	yal	281	0.000162	doğ	205	0.000118
bur	375	0.000216	sat	278	0.000160	bey	205	0.000118
bır	375	0.000216	?	276	0.000159	j	196	0.000113
koy	374	0.000216	gı	274	0.000158	1	171	9.87E-05
dav	374	0.000216	gül	273	0.000158	g	170	9.81E-05
zor	370	0.000213	çi	270	0.000156	0	163	9.41E-05
bit	366	0.000211	:	270	0.000156	!	135	7.79E-05
sen	364	0.000210	ku	267	0.000154	9	103	5.94E-05
mız	364	0.000210	eğ	266	0.000153	2	86	4.96E-05
luk	363	0.000209	sık	265	0.000153	5	76	4.39E-05
vur	360	0.000208	yin	263	0.000152	(	54	3.12E-05
bek	359	0.000207	elim	262	0.000151	)	54	3.12E-05
mala	357	0.000206	öd	259	0.000149	;	49	2.83E-05
kay	357	0.000206	cıl	259	0.000149	4	49	2.83E-05
kam	355	0.000205	çev	256	0.000148	8	47	2.71E-05
üç	354	0.000204	suç	255	0.000147	3	46	2.65E-05
gin	351	0.000203	duk	255	0.000147	c	43	2.48E-05
kol	349	0.000201	sel	254	0.000147	ğ	43	2.48E-05
gec	348	0.000201	kom	254	0.000147	7	42	2.42E-05
av	347	0.000200	kı	254	0.000147	6	31	1.79E-05
dış	345	0.000199	ucu	252	0.000145	=	3	1.73E-06
kır	341	0.000197	tüm	248	0.000143	x	2	1.15E-06
ayr	338	0.000195	yat	245	0.000141	[	2	1.15E-06
tel	331	0.000191	dağ	245	0.000141	]	2	1.15E-06

Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık
%	1	5.77E-07	}	1	5.77E-07	>	1	5.77E-07
\	1	5.77E-07		1	5.77E-07	@	1	5.77E-07
{	1	5.77E-07	<	1	5.77E-07	/	1	5.77E-07
+	1	5.77E-07	q	1	5.77E-07	#	1	5.77E-07
&	1	5.77E-07	$	1	5.77E-07			

EK 9 IHeDSH Yönteminde Kullanılan Statik Şablon

Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık
#32	978853	0.484034	dan	4491	0.002221	sı	2354	0.001164
#13	63617	0.031458	du	4430	0.002191	gel	2287	0.001131
n	28531	0.014108	ki	4411	0.002181	ır	2255	0.001115
a	25522	0.012620	ş	4403	0.002177	z	2234	0.001105
i	24985	0.012355	den	4391	0.002171	mi	2231	0.001103
y	20956	0.010363	lı	4348	0.002150	nin	2207	0.001091
e	20246	0.010011	sa	4329	0.002141	bun	2200	0.001088
ı	17105	0.008458	si	4268	0.002110	tan	2193	0.001084
r	14260	0.007051	ra	4238	0.002096	kon	2193	0.001084
an	13280	0.006567	al	4184	0.002069	değ	2186	0.001081
ğ	13172	0.006513	baş	4128	0.002041	arı	2165	0.001071
in	13040	0.006448	ay	4063	0.002009	ger	2164	0.001070
bir	12640	0.006250	s	4042	0.001999	re	2160	0.001068
ın	11939	0.005904	ün	3859	0.001908	lan	2149	0.001063
da	11828	0.005849	ol	3788	0.001873	iyor	2140	0.001058
de	10690	0.005286	at	3583	0.001772	ala	2124	0.001050
t	9774	0.004833	ıl	3508	0.001735	çık	2052	0.001015
IB	9367	0.004632	te	3474	0.001718	yıl	2048	0.001013
di	9360	0.004628	f	3444	0.001703	ıyor	2046	0.001012
le	8518	0.004212	yap	3370	0.001666	v	2045	0.001011
u	8311	0.004110	ola	3298	0.001631	ur	2027	0.001002
et	8207	0.004058	ek	3296	0.001630	gün	2027	0.001002
en	7930	0.003921	ak	3283	0.001623	par	2020	0.000999
b	7853	0.003883	tür	3098	0.001532	ası	2015	0.000996
ar	7775	0.003845	iş	3058	0.001512	kal	2003	0.000990
lar	7692	0.003804	ama	3051	0.001509	ış	2000	0.000989
ları	7390	0.003654	,	3007	0.001487	am	1990	0.000984
ma	7281	0.003600	gör	2981	0.001474	ili	1981	0.000980
o	6919	0.003421	ken	2973	0.001470	kan	1971	0.000975
il	6629	0.003278	p	2942	0.001455	ul	1970	0.000974
k	6581	0.003254	sin	2912	0.001440	yan	1951	0.000965
dı	6412	0.003171	ce	2907	0.001437	ik	1943	0.000961
ti	6373	0.003151	ka	2858	0.001413	tar	1913	0.000946
bu	6090	0.003011	sın	2844	0.001406	'	1905	0.000942
la_	5957	0.002946	kar	2805	0.001387	kur	1896	0.000938
leri	5827	0.002881	ü	2773	0.001371	var	1890	0.000935
ve	5460	0.002700	ım	2770	0.001370	dır	1889	0.000934
ler	5404	0.002672	ta	2764	0.001367	biz	1876	0.000928
li	5262	0.002602	içi	2710	0.001340	dir	1834	0.000907
m	5114	0.002529	um	2702	0.001336	alı	1832	0.000906
un	5088	0.002516	mış	2677	0.001324	diy	1809	0.000895
.	5080	0.002512	bil	2626	0.001299	ca	1795	0.000888
me	4917	0.002431	ver	2609	0.001290	kad	1793	0.000887
ir	4915	0.002430	ke	2604	0.001288	nın	1745	0.000863
er	4850	0.002398	iz	2522	0.001247	ne	1739	0.000860
im	4792	0.002370	se	2511	0.001242	lik	1738	0.000859
l	4700	0.002324	el	2483	0.001228	dah	1682	0.000832
h	4665	0.002307	miş	2472	0.001222	lu	1681	0.000831
yor	4563	0.002256	is	2464	0.001218	dem	1668	0.000825
tı	4561	0.002255	son	2449	0.001211	mak	1659	0.000820

Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık
ter	1653	0.000817	iki	1174	0.000581	ıp	845	0.000418
bak	1640	0.000811	bul	1174	0.000581	yol	842	0.000416
gib	1635	0.000808	hak	1168	0.000578	ılı	835	0.000413
or	1574	0.000778	kim	1163	0.000575	d	834	0.000412
her	1563	0.000773	esi	1160	0.000574	kap	833	0.000412
çek	1555	0.000769	uk	1153	0.000570	cak	832	0.000411
çok	1542	0.000763	su	1153	0.000570	ya	827	0.000409
mı	1534	0.000759	hiç	1153	0.000570	ci	821	0.000406
üm	1527	0.000755	ecek	1153	0.000570	siy	817	0.000404
eri	1522	0.000753	tir	1139	0.000563	tık	812	0.000402
lık	1501	0.000742	ip	1139	0.000563	laş	812	0.000402
dur	1498	0.000741	hal	1139	0.000563	ık	812	0.000402
"	1483	0.000733	on	1122	0.000555	ned	810	0.000401
aş	1481	0.000732	böy	1097	0.000542	tek	807	0.000399
bel	1472	0.000728	muş	1086	0.000537	zam	801	0.000396
ara	1471	0.000727	say	1085	0.000537	cum	796	0.000394
sor	1453	0.000718	yüz	1081	0.000535	ana	793	0.000392
man	1434	0.000709	ür	1080	0.000534	mes	792	0.000392
açı	1420	0.000702	söy	1073	0.000531	yen	791	0.000391
ey	1389	0.000687	gi	1068	0.000528	seç	782	0.000387
acak	1383	0.000684	tem	1065	0.000527	sız	773	0.000382
geç	1378	0.000681	onu	1063	0.000526	sal	769	0.000380
ül	1368	0.000676	ç	1061	0.000525	yön	765	0.000378
em	1363	0.000674	kü	1053	0.000521	dol	765	0.000378
ev	1347	0.000666	doğ	1050	0.000519	bas	754	0.000373
gen	1345	0.000665	siz	1044	0.000516	oku	753	0.000372
ile	1338	0.000662	tü	1031	0.000510	ulu	751	0.000371
düş	1318	0.000652	uş	1026	0.000507	ör	751	0.000371
yer	1307	0.000646	şey	1026	0.000507	hem	749	0.000370
maz	1306	0.000646	gaz	1012	0.000500	öne	743	0.000367
dev	1305	0.000645	lam	1008	0.000498	hay	741	0.000366
uz	1303	0.000644	şim	979	0.000484	daki	741	0.000366
yaz	1292	0.000639	büy	952	0.000471	uyor	739	0.000365
mil	1292	0.000639	lama	936	0.000463	kın	734	0.000363
tu	1290	0.000638	az	935	0.000462	duy	723	0.000358
ele	1289	0.000637	dik	931	0.000460	kaç	720	0.000356
edi	1282	0.000634	gil	921	0.000455	tik	718	0.000355
der	1273	0.000629	top	918	0.000454	üs	717	0.000355
tay	1256	0.000621	men	918	0.000454	eme	706	0.000349
lü	1255	0.000621	eği	915	0.000452	çün	698	0.000345
ben	1235	0.000611	ede	912	0.000451	öze	694	0.000343
olu	1231	0.000609	eli	910	0.000450	mah	694	0.000343
ız	1230	0.000608	hük	904	0.000447	çal	691	0.000342
mek	1222	0.000604	ada	894	0.000442	san	690	0.000341
ön	1214	0.000600	söz	892	0.000441	tur	689	0.000341
rak	1212	0.000599	yaş	890	0.000440	ded	687	0.000340
len	1209	0.000598	sür	887	0.000439	iyi	665	0.000329
yok	1194	0.000590	dün	875	0.000433	kul	660	0.000326
ük	1193	0.000590	ten	871	0.000431	gid	657	0.000325
sun	1188	0.000587	dön	865	0.000428	ağı	655	0.000324
yar	1184	0.000585	ö	852	0.000421	get	654	0.000323
tır	1181	0.000584	yak	845	0.000418	han	651	0.000322

Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık
sav	650	0.000321	çil	529	0.000262	tel	430	0.000213
deki	648	0.000320	av	529	0.000262	kab	426	0.000211
üş	646	0.000319	şu	526	0.000260	müş	423	0.000209
cı	644	0.000318	kiş	523	0.000259	mar	420	0.000208
gir	642	0.000317	sır	522	0.000258	cez	419	0.000207
daş	636	0.000314	bey	521	0.000258	bit	417	0.000206
baz	633	0.000313	üyor	520	0.000257	ref	415	0.000205
kil	632	0.000313	lem	518	0.000256	öz	408	0.000202
bin	623	0.000308	gös	510	0.000252	erek	405	0.000200
mu	621	0.000307	ah	509	0.000252	sah	404	0.000200
göz	621	0.000307	cil	504	0.000249	öl	404	0.000200
kam	620	0.000307	mer	503	0.000249	g	403	0.000199
tam	617	0.000305	kız	501	0.000248	bır	403	0.000199
kı	614	0.000304	sağ	499	0.000247	niz	401	0.000198
yet	613	0.000303	hat	497	0.000246	ucu	400	0.000198
leme	611	0.000302	adı	489	0.000242	arak	400	0.000198
kay	609	0.000301	sar	487	0.000241	haz	399	0.000197
bağ	606	0.000300	öy	486	0.000240	yür	397	0.000196
nun	602	0.000298	yay	485	0.000240	ye	397	0.000196
as	601	0.000297	dış	485	0.000240	cek	396	0.000196
oy	599	0.000296	mala	484	0.000239	boy	396	0.000196
üze	598	0.000296	kes	482	0.000238	sak	393	0.000194
mez	597	0.000295	ga	478	0.000236	ata	393	0.000194
gül	597	0.000295	ok	477	0.000236	ih	391	0.000193
tut	593	0.000293	dar	476	0.000235	vur	390	0.000193
ge	589	0.000291	zor	471	0.000233	es	388	0.000192
bay	589	0.000291	sel	470	0.000232	ağ	386	0.000191
miz	586	0.000290	tak	469	0.000232	ab	386	0.000191
büt	586	0.000290	up	467	0.000231	çi	381	0.000188
kor	584	0.000289	nas	466	0.000230	gön	380	0.000188
ban	584	0.000289	taş	464	0.000229	pek	376	0.000186
düz	580	0.000287	kom	464	0.000229	önü	376	0.000186
sev	574	0.000284	mec	463	0.000229	kaz	375	0.000185
bur	574	0.000284	kır	463	0.000229	kav	375	0.000185
dür	567	0.000280	gin	463	0.000229	ici	374	0.000185
öğ	563	0.000278	öte	462	0.000228	ses	373	0.000184
yas	560	0.000277	vey	460	0.000227	yal	371	0.000183
dü	559	0.000276	luk	457	0.000226	uza	371	0.000183
git	558	0.000276	gul	455	0.000225	mal	370	0.000183
ora	554	0.000274	faz	450	0.000223	cıl	369	0.000182
mız	554	0.000274	pol	449	0.000222	kol	364	0.000180
işi	554	0.000274	çe	448	0.000222	suç	361	0.000179
nız	552	0.000273	j	447	0.000221	ise	361	0.000179
dık	552	0.000273	ku	444	0.000220	gec	360	0.000178
hep	551	0.000272	hab	444	0.00022	yin	358	0.000177
uy	549	0.000271	sil	443	0.000219	yük	356	0.000176
sen	547	0.000270	can	439	0.000217	böl	355	0.000176
sad	543	0.000269	har	438	0.000217	isi	354	0.000175
bug	537	0.000266	dav	437	0.000216	ser	349	0.000173
leş	533	0.000264	koy	434	0.000215	mem	349	0.000173
kat	532	0.000263	ner	431	0.000213	muh	345	0.000171
din	532	0.000263	bek	431	0.000213	kit	345	0.000171

Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık	Sembol	Sıklık	Olasılık
nci	344	0.000170	gı	305	0.000151	2	86	4.25E-05
aç	340	0.000168	kut	303	0.000150	5	76	3.76E-05
tab	334	0.000165	bat	303	0.000150	c	62	3.07E-05
sek	334	0.000165	*	302	0.000149	)	54	2.67E-05
lak	332	0.000164	teş	301	0.000149	(	54	2.67E-05
ina	332	0.000164	sat	301	0.000149	;	49	2.42E-05
madan	331	0.000164	res	301	0.000149	4	49	2.42E-05
ona	330	0.000163	müc	300	0.000148	8	47	2.32E-05
iç	330	0.000163	müd	299	0.000148	3	46	2.27E-05
far	330	0.000163	mele	299	0.000148	7	42	2.08E-05
bab	330	0.000163	duk	299	0.000148	6	31	1.53E-05
gun	329	0.000163	des	295	0.000146	x	8	3.96E-06
çoc	329	0.000163	vat	293	0.000145	=	3	1.48E-06
çar	329	0.000163	HB	291	0.000144	q	2	9.89E-07
güz	328	0.000162	elim	291	0.000144	]	2	9.89E-07
emi	327	0.000162	çev	290	0.000143	[	2	9.89E-07
üç	326	0.000161	uzu	289	0.000143	>	1	4.94E-07
tüm	324	0.000160	ber	288	0.000142	<	1	4.94E-07
suz	324	0.000160	nok	286	0.000141	+	1	4.94E-07
-	322	0.000159	kür	286	0.000141	}	1	4.94E-07
id	320	0.000158	otu	285	0.000141	{	1	4.94E-07
malı	318	0.000157	mü	285	0.000141		1	4.94E-07
kin	318	0.000157	kıs	285	0.000141	\	1	4.94E-07
atı	317	0.000157	?	276	0.000136	@	1	4.94E-07
gür	314	0.000155	:	270	0.000134	/	1	4.94E-07
huk	311	0.000154	1	171	8.46E-05	&	1	4.94E-07
lük	309	0.000153	0	163	8.06E-05	%	1	4.94E-07
üz	306	0.000151	!	135	6.68E-05	$	1	4.94E-07
ıcı	306	0.000151	w	129	6.38E-05	#	1	4.94E-07
nit	305	0.000151	9	103	5.09E-05			

ÖZGEÇMİŞ

Doğum tarihi	11.02.1966	
Doğum yeri	İstanbul	
Lise	1979-1982	İst.Şehremini Lisesi
Lisans	1983-1987	Yıldız Teknik Üniversitesi Elektrik-Elektronik Fakültesi Bilgisayar Bilimleri ve Mühendisliği Bölümü
Yüksek lisans	1987-1990	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Bilimleri ve Mühendisliği Bölümü
Doktora	1992-1999	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Bilimleri ve Mühendisliği Bölümü
Çalıştığı kurum	1988-Devam ediyor	Y.T.Ü Bilgisayar Bilimleri ve Mühendisliği Bölümü'nde Araştırma Görevlisi