Sözlük Tabanlı Veri Sıkıştırma Projesi

Dictionary Based Coding Project

Berkay Efe ÖZCAN
Mühendislik Fakültesi, Bilgisayar Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye
berkayefeozcan@gmail.com

***Özetçe*—Bu projenin amacı geçmişte ve günümüzde sıklıkla kullanmakta olan sözlük tabanlı verileri sıkıştırma algoritmalarından ikisininin sıkıştırma yöntemlerinin araştırlması ,avantaj ve dezavantajlarının öğrenilip performanslarının karşılaştırılmasıdır. Projede kıyasladığmız algoritmalar LZ77 ve deflate algoritmalardır.Karşılaştırma işlemi için metin.txt girdi dosyası sıkıştırılmaktadır ve sıkıştırlmış veri her algoritma için farklı dosyalara yazılmaktadır.Bu proje C dili ile geliştirilmiştir. Kullanılan algoritmaya dair detaylar ve yöntemler ileriki bölümlerde anlatılmıştır.**

Anahtar Kelimeler—Veri sıkıştırma, Sözlük tabanlı kodlama, Deflate, LZ77,Huffman kodlaması,LZSS.

Abstract — The aim of this project is to investigate the compression methods of two of the dictionary-based data compression algorithms that have been used frequently in the past and today, to learn their advantages and disadvantages and to compare their performance. The algorithms we compare in the project are LZ77 and deflate algorithms. For comparison, the text.txt input file is compressed and the compressed data is written to different files for each algorithm. This project is developed in C language. Details and methods of the algorithm used are described in the next sections.

Keywords— Data Compression, Dictionary based coding, Deflate, Lz77,Huffman coding,LZSS.

I. GİRİŞ

Bilgisayar biliminde, veri sıkıştırma, bir verinin önceden belirlenmiş olan bir kodlama şeması ile daha az depolama alanı kullanılarak tamamen veya kısmen ifade edilmesidir.Veri sıkıştırma işlemleri fiziksel ortamdaki saklama yerlerinden tasarrufu sağlamak ,veri iletişim ağlarında verinin hızla iletilebilmesi gibi pekçok alanda kullanılmaktadır. Veri sıkıştırma algoritmaları kayıplı veri sıkıştırma ve kayıpsız veri sıkıştırma algoritması olarak ikiye ayrılır.Verinin sıkıştırlırken kaybolduğu algoritmalara kayıplı veri sıkıştırma algoritmaları denir.Bu nedenle tüm bilgilerin önemli olduğu dosyalarada tercih edilmezler. Sıkıştırmanın temelinde tekrar eden verilerin bir veriymiş gibi temsil edilmesi vardır. Kayıplı sıkıştırma algoritmalarında ise bu temsil etme işlemi veriden kayıp olmasına sebep olabilir. Kayıpsız sıkıştırma yöntemi ise iki tanedir.Bunların ilki değişken uzunluklu kodlama olarak ta bilinen olasılık tabanlı kodlama,ikincisi ise sözlük tabanlı kodlamadır.Olasılık tabanlı kodlamada sıkıştırma işlemi,sıkıştırılacak verinin bütününde tekrar eden kısımları sıkıştımadan önceki bit sayısına nazaran daha az bit ile ifade edilmesi prensibi ile sıkıştırılmaktadır.Olasılık tabanlı kodlamada çok kullanılan teknikler ***Huffman Kodlaması*** ve ***Aritmatik Kodlama*** 'dır. Sözlük tabanlı kodlamada da tekrar eden karakter katarları tek bir sembol grubuyla ifade edilir.En çok kullanılan sözlük sıkıştırma teknikleri LZ77,LZ78 ve LZW dir.

Bu projede ise sözlük tabanlı kayıpsız veri sıkıştırma algoritmalarından LZ77 ve Deflate algoritmalarını incelendi ve performansları karşılaştırıldı. Algoritmaların detayları ileriki bölümlerde açıklanmıştır.

II. TEMEL KAVRAMLAR

i.*Entropi* :

Entropi genellikle bir sistemdeki rastgelelik ve düzensizlik (kaos) olarak tanımlanır.Sıkıştırma işlemi için entropi ne kadar az ise sıkıştırma işlemi o derece fazla olacaktır.

ii.*Arama Tamponu (Search Buffer)* :

Arama tamponu karakter tabanlı sıkıştırma algoritmalarında tek bir karakter olarak sembol edilecek verinin –bu veri ileri tampondan elde edilir -tampon içerisinde arama yapılamasını sağlamak amacıyla oluşturulur.Arama tamponu ne kadar büyük olursa sıkıştırma oranı o derece artmaktadır fakat her metin için bu arama tamponu dolaşılacağı için sıkıştırma süresi artmaktadır.Arama tamponundaki veriler kodlanmış verilerdir.

iii.İleri Tampon (Lookahead Buffer) :

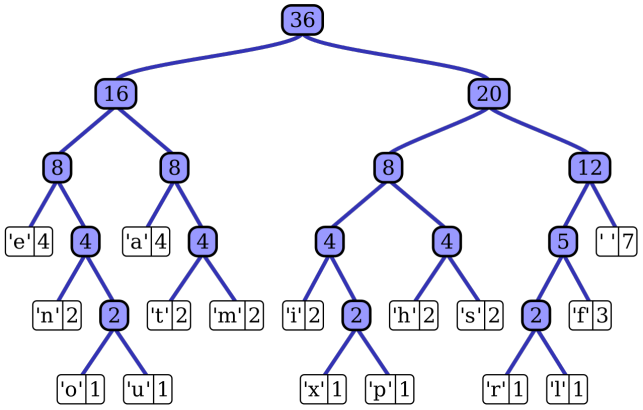
Tek bir sembol olarak kodlanacak olan karakter arama tamponu içerisinde aranır. Bu arama sonucunda uzaklığı (offset) ve eşleşmenin uzunluğu elde edilip kodlanır. İleri tamponun büyüklüğü maxsimum eşleşme uzunluğu verir. İleri tamponundaki veriler henüz kodlanmamış verilerdir.

III. KULLANILAN TEKNİKLER VE KIYASLAMASI YAPILAN ALGORİTMALAR

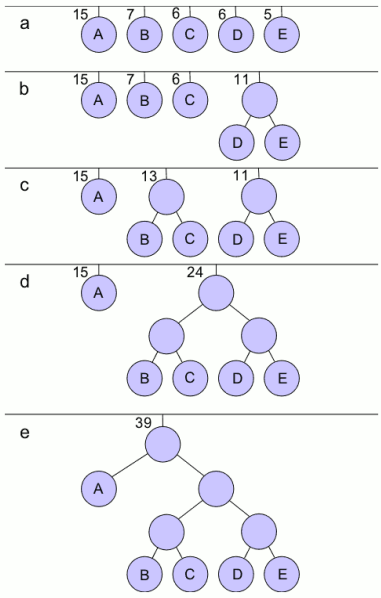
A. Huffman Kodlaması :

Huffman kodlaması kayıpsız veri sıkıştırmada kullanılan bir algoritma olup David A. Huffman tarafından geliştirilmiştir. Huffman kodlamasının en büyük avantajlarından birisi kullanılan karakterlerin frekanslarına göre bir kodlama yapması ve en çok tekrar eden karakterler en az bitle ifade edilmektedir. Bu bize entropi ne kadar az ise o oranda daha fazla sıkıştırma yapabilmeyi sağlar.

Örnek olarak karakterler ve frekansları {{‘boşluk’,7},{‘a’,4},{‘e’,4},{‘f’,3},{‘h’,2},{‘i’,2},{‘m’,2},{‘n’,2},{‘s’,2},{‘t’,2},{‘l’,1},{‘o’,1},{‘p’,1},{‘r’,1},{‘u’,1}, {‘x’,1}, } olarak verilmiş olsun. Huffman kodlaması karakterleri frekanslarına göre sıralar ve en küçük iki frekansa sahip karakteri ikili ağaca ekler. Ağacın ebeveynleri ı ağaca eklenecek karakterlerin frekanslarının toplamından oluşur ve çocukları ise karakterlerdir. Bu şekilde ağaca ekleme işlemi devam ettirilir ve algoritma ağaca eklenecek karakter kalmadığında son bulur.



Şekil 1: Yukarıdaki örneğin huffman ağacı

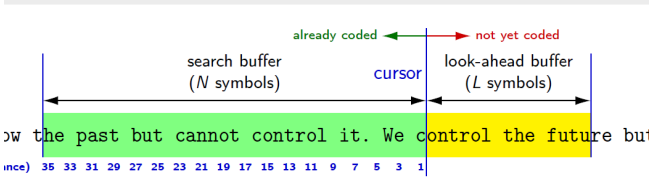


Şekil 2: Farklı bir örnek için Huffman Ağacının oluşturulma aşamalarının gösterimi

B. LZ77 Algoritması:

LZ77 algoritması, Abraham Lembel ve Jacob Ziv tarafından 1977 yılında yazdıkları A Universal Algorithm for Sequential Data Compression başlıklı makale ile ortaya atılmıştır. Bu algoritma metinlerde geçen bağlaç, zamir gibi sık sık tekrar eden karakter katarlarını tek bir sembol halinde kodlama yaparak sıkıştırma yapar.

Algoritmanın başlangıcında arama taponu ve ileri tampon metinin başındadır ve her iki tamponda boştur. İleri tampon arama tamponu içerisinde bulamadığı karakterlerin uzaklık değerini ve eşleşme uzunluğunu 0 olarak kodlanır. Bu durumda örneğin ‘c’ harfinin kodlanmasının sembolik gösterimi <0,0,c> olacaktır. Arama tamponu ileri tamponundan arama tamponunun büyüklüğü kadar uzaklığa konumlandırılır. Eğer arama tamponu metinin başlangıç adresinden küçükse metinin başlangıç adresine eşitlenir. Bu şekilde sürekli arama tamponu içerisinde ileri tampondaki karakterler aranır. Eğer eşleşme olursa eşleşmenin uzunluğu kadar ileri tampon ileriye alınır. Metinin sonuna gelindiğinde algoritma sonlanır.



Şekil 3: Arama tamponu ve ileri tamponun şematik gösterimi

Şekil 4 te verilen örneğin arama tamponu 8 bayt, ileri tamponun uzunluğu 4 bayt olarak belirlenmiştir.

Miss␣Mississippi

search buffer	(d, ℓ, n)	decoded phrase
	(1, 0, M)	M
M	(1, 0, i)	i
Mi	(1, 0, s)	s
Mis	(1, 1, ␣)	s␣
Miss␣	(5, 3, s)	Miss
iss␣Miss	(3, 3, i)	issi
Mississi	(1, 0, p)	p
ississip	(1, 1, i)	pi

Şekil 4: LZ77 nin kodlanmış örneği

C ile geliştirdiğim programda LZ77 algoritması için arama tamponun büyüklüğü 500, ileri tamponun büyüklüğünü de 50 olarak ayarladım. Bundan dolayı kodlanmamış her bir karaktere karşılık 3 bayt kodlanmış karakter denk gelmektedir. Gerçekleştirdiğim testler sonucunda LZ77 için bir hata almamakla beraber arama tamponun büyüklüğüne göre sıkıştırma süresi artmaktadır.

C. LZSS Algoritması:

Lempel-Ziv-Storer-Szymanski Algoritma'sı olarak da bilinir. LZSS'in bir yeniden yapılandırılmış halidir. LZSS algoritmasında LZ77 algoritmasında da olduğu gibi tekrar eden karakterler tek bir sembol ile gösterilir. Aralarındaki temel fark LZSS algoritması herhangi bir eşleşme bulamadığında, offset ve eşleşme uzunluğunu yazmayarak gereksiz yer işgalinin önüne geçmesidir.

Şekil 5 teki örnekte arama ve ileri tamponları Şekil 4 olduğu gibidir.

Message: Miss␣Mississippi

search buffer	look-ahead	$(b, \{d, \ell\} \mid n)$
	Miss	$(0, \mathtt{M})$
M	iss␣	$(0, \mathtt{i})$
Mi	ss␣M	$(0, \mathtt{s})$
Mis	s␣Mi	$(1, 1, 1)$
Miss	␣Mis	$(0, \mathtt{␣})$
Miss␣	Miss	$(1, 5, 4)$
iss␣Miss	issi	$(1, 3, 4)$
Mississi	ppi	$(0, \mathtt{p})$
ississip	pi	$(1, 1, 1)$
ssissipp	i	$(1, 3, 1)$

Şekil 5: LZSS sıkıştırma örneği

D. Deflate Algoritması:

Deflate algoritması kayıpsız veri sıkıştırma algoritmasıdır. Günümüzde gzip ve png dosya formatlarında kullanılmaktadır. Algoritma LZSS – veya LZ77 – algoritmasının ve Huffman kodlamasının birleşiminden oluşmaktadır. LZSS ile sıkıştırılan veriler huffman kodlamasıyla daha da sıkıştırılır. Deflate algoritması bloklar halinde çalışır. Her bloğun başına huffman ağacı yazılır. Ardından da kodlanmış veri eklenir. Blok blok işlem yapıldığında daha hızlı sıkıştırma yapılmaktadır. Fakat blok sayısı arttıkça tüm metin için bir karakterin tekrar sayısı azalacağı için huffman kodlaması daha fazla yer işgal edip algoritmanın verimliliğini düşürecektir.

Deflate algoritmasını koda dökerken Şekil 2 deki Huffan ağacının aşamalarını yapabilmek için bağlı liste kullandım. Böylece karakterleri frekanslarına göre sıralayıp link liste ekledim. Huffman kodlaması sonucu oluşan kodları ise binary search tree ye ekleyerek daha hızlı bir şekilde her karakter için oluşturulan kodlara ulaştım. Ayrıca LZSS algoritmasındaki arama tamponunu 32 000 bayt olarak, ileri tamponun uzunluğunu ise 255 bayt olarak belirledim. Deflate algoritması çalışırken hata oluşabilmekte bu hata ileriki bölümlerde detaylı olarak anlatılacaktır.

LZSS algoritmasında bayrak ve offset i tutması için 16 bitlik bir uint8_t tipinde bir değişken tanımladım. Bunun 1 biti bayrak için 15 biti de offset için ayırdım. En yüksek öncelikli bit bayrak için ayrıldı. Bu işlemleri yaparken bit kaydırma , 'veya' lama, bit maskeleme gibi yapılar kullandım.

```
#define FLAGBITS 1
#define SS_OFFSETBITS 15
#define SS_OFFSETMASK ((1 << (SS_OFFSETBITS)) - 1)
#define SS_GETFLAG(x) (x >> SS_OFFSETBITS)
#define SS_GETOFFSET(x) (x & SS_OFFSETMASK)
#define SETFLAGANDOFFSET(x,y) ((x <<SS_OFFSETBITS) | y)
```

Şekil 6 : Bayrak ve offset için bit işlemlerini yapan tanımlamalar

Karakter ve eşleşme uzunluğunu ise 8 bitlik uint8_t tipinde değişkende tuttum. Eğer eşleşme varsa bayrak değeri 1 oluyor ve eşleşme değeri uint8_t tipindeki değişkene atanıyor. Halihazırda bayrak 1 olduğunda geriye kalan 15 bit değişkene de offset atanır. Eğer bayrak sıfırsa 8 bitlik değişkene karakterin ascii tablosundaki decimal sayı atanmaktadır. Şekil 7 de bu değişkenleri tutan yapı gösterilmiştir.

```
struct lzSSToken
{
    uint16_t flagAndOffset; //
    uint8_t lengthOrCharacter;
};
```

Şekil 7: Bayrak , offset, eşleşme uzunluğu veya karakteri tutan yapı.

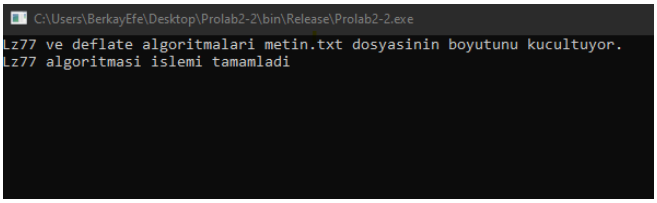
Şekil 8 de ise LZ77 algoritması için tanımlanan yapı konuyla ilişkisi sebebiyle buraya alınmıştır.

```
struct token
{
    // bu yapi lz77 i
    uint8_t offset;
    uint8_t length; //
    // eger uzungugu
    char character; //
};
```

Şekil 8 : LZSS in tokeni için tanımlanan yapı

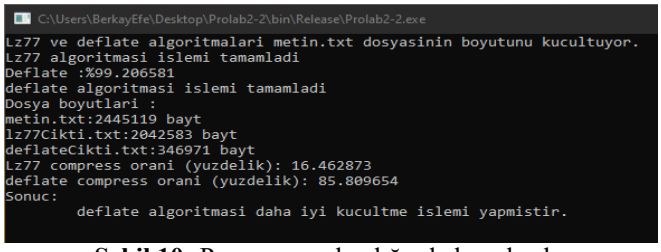
IV. PROGRAMIN ÇALIŞTIRILMASI

Program exe dosyası üzerinden çalıştırılabilir. Program çalıştırıldığında eğer metin.txt dosyası açılabiliyorsa Şekil 9daki ekranla bizi karşılar. metin.txt dosyası sıkıştırılacak olan dosyadır. Program metin.txt binary olarak okur ve sıkıştırma sonuçlarını da binary olarak dosyalara yazılır.



Şekil 9 : Program çalıştırılınca karşılaşılan konsol ekranı

Sırasıyla LZ77 ve deflate algoritmaları çalışmaya başlar. Algoritmalar sıkıştırma işlemini tamamladığında sıkıştırılmış dosyalar oluşur. Bu dosyalar 'lz77Cikti.txt' ve 'deflateCikti.txt'dir. Program hatasız sonlanıyorsa bizi şekil 10daki konsol ekranı karşılar.



Şekil 10: Program sonlandığında karşılaşılan konsol ekranı

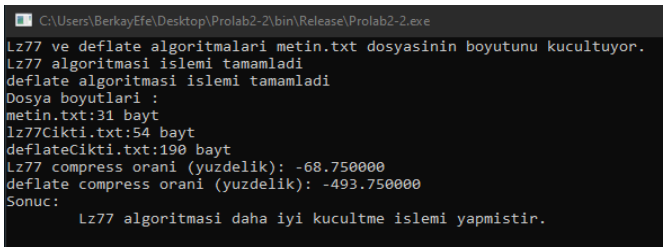
V. ÇALIŞMA ZAMANINDA ORTAYA ÇIKABİLECEK HATALAR

LZ77 algoritması küçük boyutlu dosyalarda da büyük boyutlu dosyalarda da hata vermeden çalışabilmektedir. Deflate algoritması 100 mb gibi boyuta sahip dosyaları sıkıştırmak için çalıştığında Segmentation fault hatası verebilmektedir. Bu hataya çözüm bulamadım. Bu hata dışında çıktı dosyalarının da açılamaması da programın hatalı sonlanır. Programın hata oluşmadığında dönüş değeri 0 dır.(Return 0)

VI. ALGORİTMALARIN KARŞILAŞTIRILMASI VE GERÇEKLEŞTİRİLEN TESTLER

Proje için geliştirilen program 1mb ,10 mb ,1kb boyutlara sahip metin dosyalarını sıkıştırarak test edildi. Düşük boyutlu dosyaları LZ77 algoritmasının Deflate algoritmasına nazaran daha iyi sıkıştırma yaptığı, buna karşıt büyük boyutlu metin dosyalarında ise Deflate algoritması daha iyi sıkıştırma yaptığı gözlemlenmiştir. Teorik olarak da küçük boyutlu dosyalarda LZ77 algoritmasının, büyük boyutlu dosyalarda ise Deflate algoritmasının daha iyi sıkıştırma yapacağı hesaplanmış ve ön görülmüştür. Çünkü küçük boyutlu dosyalarda entropi fazla olabileceğinden lz77 bir karakter başına daha az baytla sembolize eder. Ama Deflate algoritması ise dosyaya blok başına huffman kod ağacını da yazdığından daha verimsiz bir sıkıştırma yapar. Büyük boyutlu metin dosyalarında ise karakter tekrarı daha fazla olduğundan huffman kodlaması daha efektif çalışarak daha iyi bir sıkıştırma performansı sağlar.

Ayrıca küçük boyutlu bazı dosyalarda her iki algoritmada da sıkıştırılmış dosyalar sıkıştırılacak dosyadan daha büyük boyuta sahip olabilir. Bir başka deyişle dosyada sıkıştırma değil genişleme olmuştur. Bu durum şekil 11 de gösterilmiştir.



Şekil 11: 31 baytlık bir dosyanın sıkıştırılınca karşılaşılan konsol ekranı

VII. ÖZET VE SONUÇLAR

Bu projede dosya sıkıştırma algoritmalarından LZ77 ve Deflate algoritmaları karşılaştırıldı ve performansları test edildi. Veri sıkıştırma hakkında temel bilgiler edinildi. Bazen Deflate bazen de LZ77 algoritmasının daha iyi sıkıştırma yaptığı gözlemlendi. C dilinde bellek yönetimi ,binary search tree ,link list gibi konular pekiştirildi. Sıkıştırma algoritmaları implement edilip dosya sıkıştırma hakkında fikir edinildi. Segmentation fault hakkında araştırma yapıp sebeplerini öğrenildi.

Veri sıkıştırma verinin bir yerden bir yere aktarımı ve depolama alanlarında önemli bir yere sahip olduğu, bazı sıkıştırma algoritmalarının patentle korunduğu bu proje kapsamında öğrenildi. Sonuç olarak sıkıştırma algoritmalarının kullanım yerlerine göre avantajları ve dezavantajları olduğu gözlemdi ve arama tamponunun boyut artışı sıkıştırma performansını arttırdığı fark edildi.

VIII. KAYNAKÇA

[1] https://en.wikipedia.org/wiki/DEFLATE

[2]https://www2.cs.duke.edu/courses/spring03/cps296.5/papers/ziv_lempel_1977_universal_algorithm.pdf

[3] http://bilgisayarkavramlari.sadievrenseker.com/2009/02/25/huffman-kodlamasi-huffman-encoding/

[4] https://en.wikipedia.org/wiki/Segmentation_fault

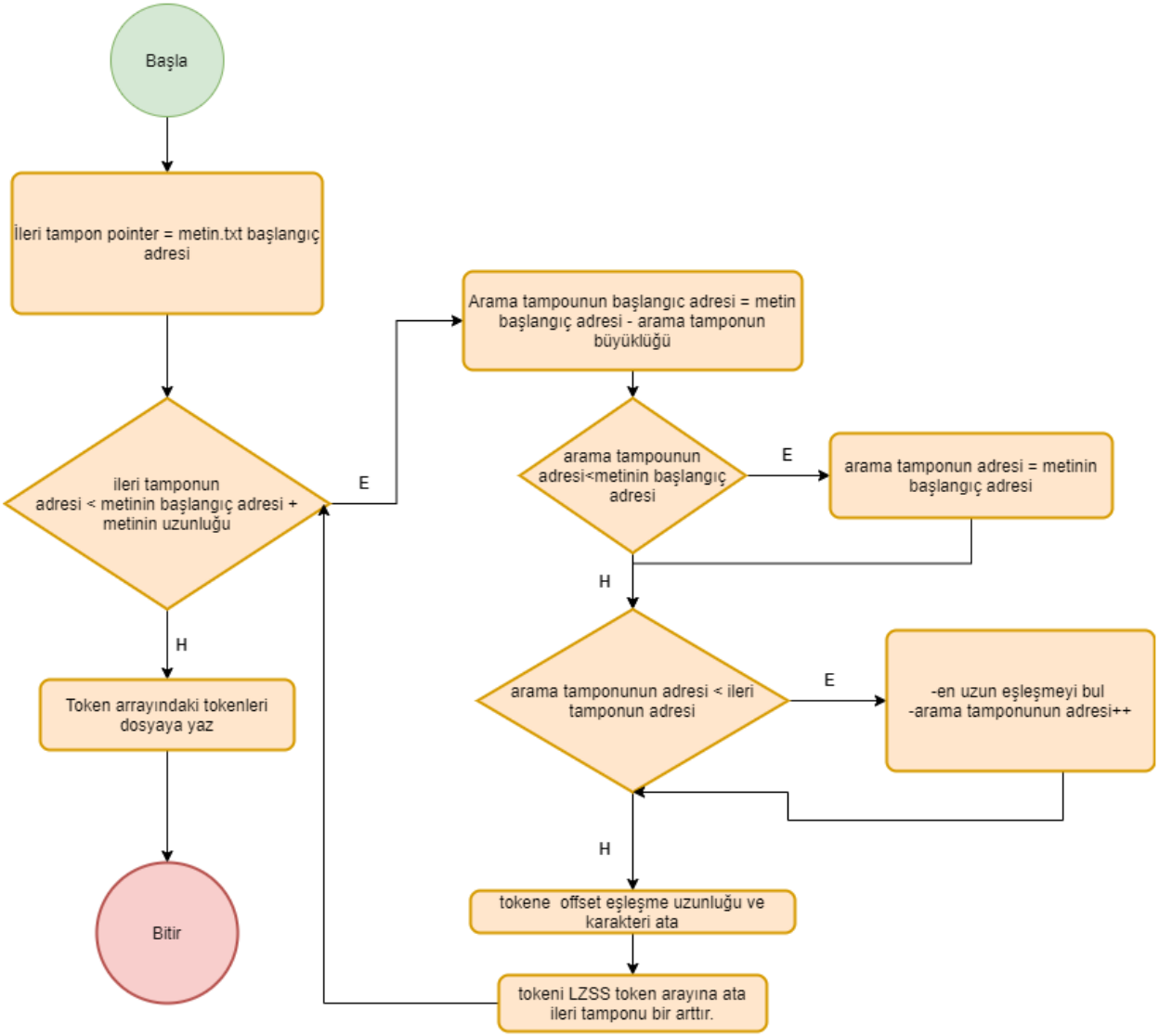
[5] https://ysar.net/algoritma/lz77.html

[6] https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/

[7] https://e-bergi.com/y/veri-sikistirma/

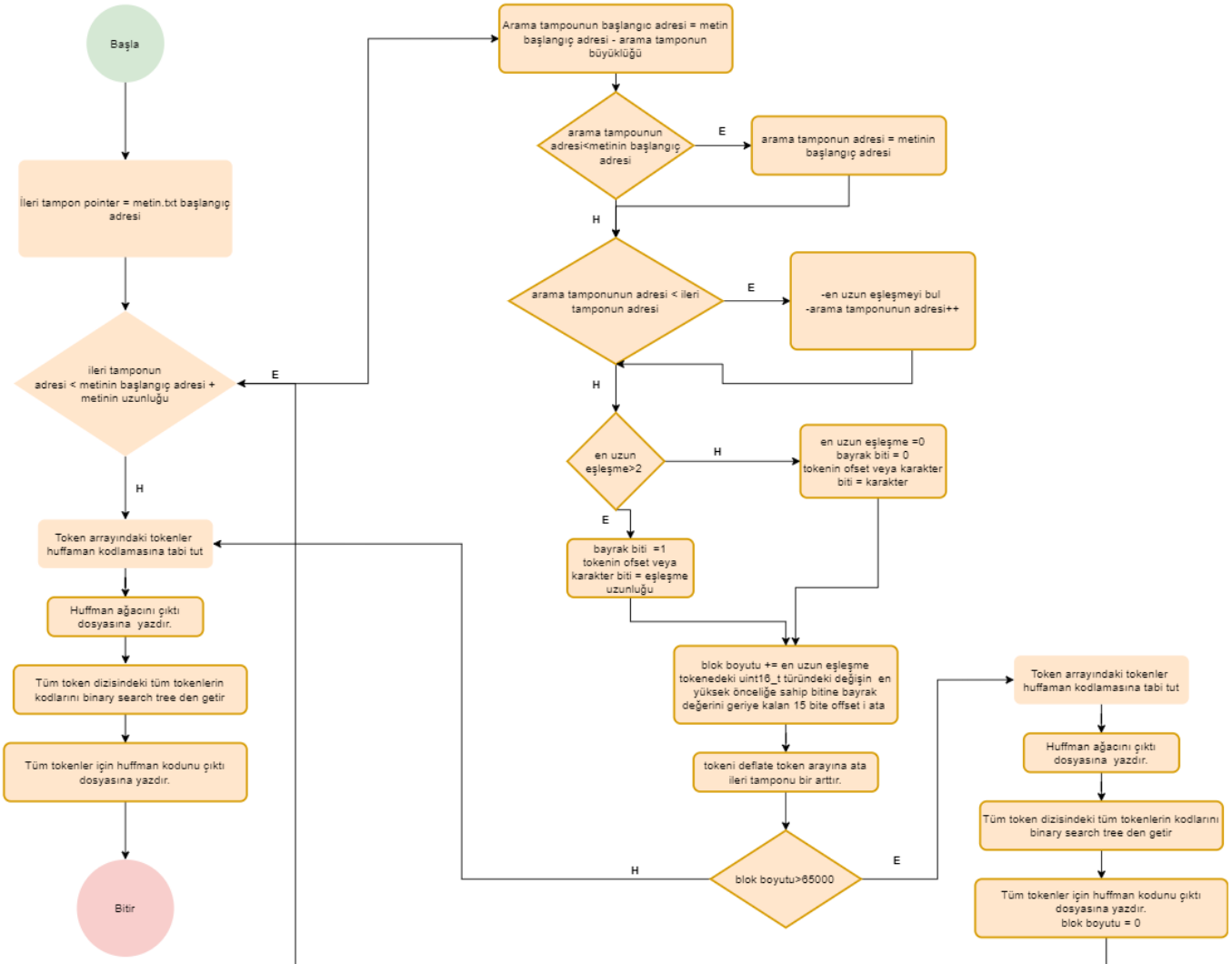
[8] https://www.cs.cmu.edu/~adamchik/15-121/lectures/Linked%20Lists/linked%20lists.html

IX. LZ77 ALGORİTMASININ AKIŞ ŞEMASI



Şekil 12

X. DEFLATE ALGORİTMASININ AKIŞ ŞEMASI



Şekil 13