LZ77 VE DEFLATE SIKIŞTIRMA METOTLARI

Programlama Laboratuvarı II
BLM 210

Hilal Özkan 180201017
Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi
hilal_ozkan1200@hotmail.com

Edmond Vujici 170201127
Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi
edmondvujic6@gmail.com

ABSTRACT – ÖZET

Bu projede karakter tabanlı LZ77, DEFLATE sıkıştırma algoritmaları kullanılarak dosya sıkıştırma uygulaması yapmamız istenmektedir. Veri sıkıştırma yöntemleri, sıkıştırma biçimlerine göre kayıplı ve kayıpsız sıkıştırma olarak iki grup altında incelenebilir. Kayıplı sıkıştırma yöntemlerinde, verilerin sıkıştırıldıktan sonra, orijinal verinin kısmen kaydedildiği ve tekrar açılmak istendiğinde eski verinin tam olarak elde edilemediği yöntemlerdir. Kayıpsız sıkıştırma yöntemlerinde ise verinin orijinal haline ulaşmak mümkündür.

I. ANAHTAR KELİMELER

Sıkıştırma yöntemleri, Sözlük tabanlı sıkıştırma, LZ77 algoritması, LZSS algoritması, Huffman algoritması, DEFLATE algoritması.

II. GİRİŞ

Bu proje kapsamında LZ77 ve DEFLATE algoritmaları kullanarak sözlük tabanlı sıkıştırma yapmamız istenmiştir.

Olasılık tabanlı kodlamada, sıkıştırılan verinin bütünü içinde daha sık kullanılan sembollere daha küçük boyutta bitlerden oluşan kodlar atanması prensibi ile sıkıştırma yapılır. En çok kullanılan olasılık tabanlı teknik ise Huffman kodlamasıdır.

Sözlük tabanlı kodlama ise sık tekrarlanan sembol grupları için tek bir sembol kullanılması ile sıkıştırma yapılır. En çok kullanılan sözlük tabanlı teknikler, LZ77, LZ78, LZW ve LZSS yöntemleridir. Hem olasılık tabanlı kodlamayı hem de sözlük tabanlı kodlamayı bir arada kullanan algoritmalar, daha yüksek sıkıştırma oranları sağlar.

DEFLATE algoritması bunun bir örneğidir. DEFLATE algoritması Huffman ve LZ77 sıkıştırma tekniklerinin bir arada kullanan bir algoritmadır.

III. TEMEL KAVRAMLAR

Proje gelişiminde: Programlama dili olarak Standart C kullanılmıştır. Geliştirme ortamı olarak Code::Bloks 20.03 kullanılmıştır. Derleyici olarak GNU GCC kullanılmıştır. Projeyi çalıştırmak için projenin klasörüne kaynak.txt dosyasını ekleyiniz ve Code:: Bloks'u çalıştırın. Programı run butonuna basarak çalıştırınız.

IV. YALANCI KOD

A. prosedür LZ77 kodlanmış(string metin, INT limit, INT belirtec_sayisi)

- bel_say INT tipinde tanımla ve sıfıra eşitle.
- INT tipinde kapasiteyi tanımladıktan sonra üç bit kaydır.
- belirteç oluştur.
- ileri ve arama tampon string'leri oluştur.
- kodla belirteci oluştur ve kapasite kadar bellek ayırt.
- ileri tampon metin ve limitin toplamından daha küçük iken ileri tampona metini ata ve ileri tamponu gez.
  - arama tampona kontrol edilen ileri tampon biti ata.
  - eğer arama tampon metinden daha küçük ise
    - arama tampona metini ata.
  - eşleşme boyutu tanımla ve eşleşmeye ileri tamponunu at.
  - arama tamponu ileri tampondan daha küçük iken arama tamponunu gez.
    - eşleşme uzunluğu bul.
    - eğer eşleşme uzunluğu eşleşme boyutundan daha büyük ise
      - eşleşme boyuta uzunluğu ata.
      - eşleşmeye arama tamponu ata.
  - eğer eşleşmenin son karakteri dahil ise eşleşmeyi kısalt.
  - ileri taponu yazdır.
  - kaydırma miktarı belirt.
  - bulunan benzetmeyi yazdır.
  - ileri tampona eşleşme boyutu ata.
  - eğer belirteç sayısı kapasiteden daha büyük ise kapasiteyi arttır.
- kodlanmış metini oluştur ve geri döndür.

B. **prosedür Huffman'ı çalıştır(string harfler, INT frekansları, INT karakter sayısı, string orijinal cümle)**

- yığının sıfırıncı elemanı için yer ayır.
- yığının sıfırıncı elemanına 0 frekansı belirt.
- karakterleri gezerken
  - her karakter için geçici bir düğüm oluştur.
  - geçici düğüme i'ninci harfi ve frekansı ata.
  - geçici düğümünü solu ve sağı boş olarak işaret et.
  - geçici düğümü ağaca ekle (eleman ekleme fonksiyonu).
- eğer eleman sayısı bir ise
  - karakterin frekansı sıfır olarak belirt.
- karakter sayısının bir eksisine kadar gezerken
  - sol düğümünün yığını sil.
  - sağ düğümünün yığını sil.
  - geçici düğüm için yer ayır.
  - geçici düğümün harfi sıfır yap.
  - geçici düğümün soluna solu ata.
  - geçici düğümün sağına sağı ata.
  - geçici düğümün frekansı solun ve sağın frekansının toplamına eşitle.
  - geçici elemanı ağaca ekle (eleman ekleme fonksiyonu).
- ağaç düğümü oluştur ve yığını sil.
- kod string'i tanımla.
- kod string'i boşalt.
- Huffman Ağacı yazdır.
- Huffman Cümlesini yazdır.

C. **prosedür eleman ekleme(düğüm tipinde eleman)**

- yığını büyüklüğü arttır.
- yığına bir eleman ekle.
- şimdiki yığın büyüklüğu al.
- yığının şimdiki elemanın yarısının frekansı elemanın frekansından daha büyük iken
  - yığının şimdiki elemana yığının şimdiki elemanın indisinin yarısındaki elemanı ata.
  - şimdiki yığının büyüklüğünün yarısını al.
- yığının şimdiki indisine elemanı ata.

D. **prosedür frekanslar (string str, INT blok sayısı, FILE * dosya)**

- sayaç'a sıfır ata.
- frekanslar dizisi oluştur.
  - str'yi gezerken frekanslar dizisinin str'inci elmanı arttır.
    - i, 256'dan daha küçükken eğer frekansların
    - i'ninci elemanı sıfır değilse sayacı arttır.
- sayaç boyutunda frekans dizisi oluştur, sayaç boyutunda CHAR tipinde harfler dizisi oluştur.
- harfi ve ait olduğu frekansı yazdır.

E. **prosedür dosya_DEFLATE(FILE * dosya)**

- sayaca sıfır ata.
- INT tipinde LZSS için belirteç sayısı belirt.
- dosyayı aç.
- INT tipinde blok uzunluğu tanımla.
- blok uzunluğu kadar string tanımla.
- blok uzunluğu kadar buffer tanımla.
- CHAR tipinde 100 elemanlı stringler matrisi tanımla.
- dosyadan blokları blok uzunluğu kadar alırken
  - alınan blokları LZSS algoritmasıyla şifrele
  - bloğun LZSS şifrelenmesini yazdır.
  - bloğun LZSS şifrelenmesinden elde edilen harf frekanslarını yazdır.
- gönderilen frekanslara göre Huffman kodunu bul.

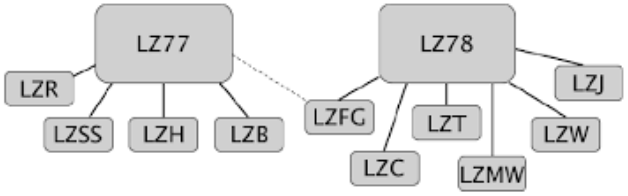
V. YÖNTEM

İlk olarak kaynak.txt dosyası içerisindeki metini okuduk ve bir CHAR dizisine atadık. Aldığımız bu metini LZ77 ve DEFLATE algoritmalarında sıkıştırarak sonuç.txt dosyasına yazdırdık.

- **LZ77 algoritması**

Abraham Lempel ve Jakob Ziv tarafından geliştirilen ve 1977 yılında yayınladıkları "A Universal Algorithm for Data Compression" isimli makalelerinde tanımladıkları bu yöntem, o yıllarda tüm dünyada büyük ilgi görmüştür. Algoritmanın eksik yönleri zaman içinde farklı bilim adamları tarafından geliştirilmiştir. Sonraları yeni geliştirilen algoritmaların hepsine LZ77 ya da LZ1 ailesi denilmiştir.

LZ77 ailesi metin tabanlı veri sıkıştırmada büyük aşama kaydedilmesinin yolunu açmış, PKZip, Zip, Lharc

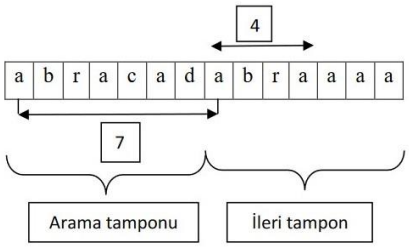


(LHA) ve ARJ gibi 80'li ve 90'lı yılların popüler sıkıştırma paketleri değişken uzunluklu kodlayıcı ile desteklenen LZ77 tabanlı algoritmalar kullanmışlardır.

LZ77 yaklaşımında sözlük, daha önce kodlanmış serinin bir parçasıdır. Algoritmadaki arama tamponunun büyüklüğü, daha önce kodlanmış serinin ne büyüklükte bir parçasında arama yapılacağını belirler. Arama tamponu büyütüldükçe, sıkıştırma oranı artar, fakat aynı zamanda sıkıştırma zamanı da artar.

❖ **Örneğin:**

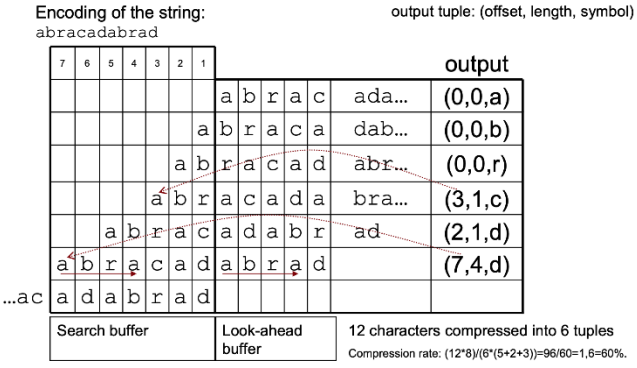
İleri tamponun ilk karakteri olan a, arama tamponunda sondan başa doğru aranır. İkinci karşılaştırmada benzerlik bulunur, fakat bu karakterden sonra b karakteri değil de d karakteri yer aldığı için benzerlik uzunluğu sadece 1'dir. Arama devam ettirilir. İki karakter sonra bir a daha bulunur, sonrasında c yer



aldığı için bunun da benzerlik uzunluğu 1'dir. Aramaya devam edilir.

Arama tamponunun başında, yani ileri tamponda aranan karakterden 7 uzaklıkta (offset=7) bir a daha bulunur. Bu defa benzerlik uzunluğu 4'tür (abra). İleri tamponda "abra" serisinden sonra yer alan a karakteri ile birlikte [7,4,C(a)] şeklinde üçlü olarak kodlanır. İleri tamponun en sonundaki a karakteri ise [0,0,C(a)] şeklinde kodlanır.

Aşağıda örnek gösterilmiştir:



Projemizi oluştururken ileri tamponun uzunluğu ve arama tamponunun uzunluğu için bazı belirteçler belirledik. Kaydırma bitini 5 bit, kaç tekrar olduğunu saklamak için 3 bit, karakter uzunluğunu saklamak için de 8 bit olarak belirledik. Bu nedenle her bir buffer toplamda 16 bit boyutunda oluştu.

Belirlediğimiz boyutlardan yola çıkarak arama bufferı'nın uzunluğu 31 bit olacak yani maksimum 31 bit geriye giderek arama yapılabilir. Eğer daha iyi bir sıkıştırma istiyorsak bu bit sayısını artırarak daha az bir boyut elde edebiliriz fakat sıkıştırma süremiz de artar.

Gene belirlediğimiz boyutları temel alarak ileri buffer'ın uzunluğu da maksimum 7 bit olabilir çünkü bir bit sonlandırma ("\0") karakterine ayrılır.

Buffer'lerın boyutlarını kaynakçada belirttiğimiz 1. maddedeki siteden elde ettiğimiz bilgiler doğrultusunda seçtik.

❖ **LZSS algoritması**

Lempel – Ziv – Storer – Szymanski (LZSS), 1982'de James Storer ve Thomas Szymanski tarafından oluşturulan, LZ77'nin bir türevi kayıpsız bir veri sıkıştırma algoritmasıdır.

❖ **LZ77 ve LZSS arasındaki fark**

LZ77 ve LZSS arasındaki temel fark, LZ77 de sözlük referansı değiştirmeye çalıştığı string'den daha uzun olabileceğidir. LZSS de karşılaştırma yaparken kırılma noktası yaşandığında bir sonraki eleman alınmaz. Yani bir sonraki karşılaştırmada ilk karşılaştırılacak eleman olarak kullanılır.

Message: `Miss␣Mississippi`

search buffer	look-ahead buffer	(d, ℓ, n)
	Miss	(1, 0, M)
M	iss␣	(1, 0, i)
Mi	ss␣M	(1, 0, s)
Mis	s␣Mi	(1, 1, ␣)
Miss␣	Miss	(5, 3, s)
iss␣Miss	issi	(3, 3, i)
Mississi	ppi	(1, 0, p)
ississip	pi	(1, 1, i)

LZ77'de üçüncü satırda göründüğü gibi 's' benzetmesi bulundu ama bir sonraki karakter de alındı. LZSS'deki durum böyle değil.

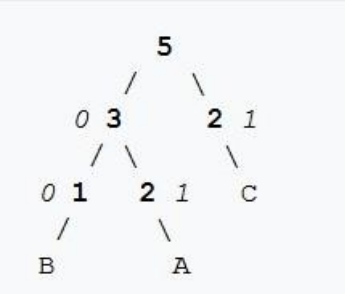
Message: `Miss␣Mississippi`

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

❖ **Huffman Kodlaması**

MIT'de Robert Fano tarafından kurulan - ve bilişim teorisi alanında bir ilk olan - sınıfta öğrenci olan David Huffman tarafından verilen bir ödev üzerine geliştirilmiştir [Huffman, 1952].

GZIP, JPEG, MPEG gibi yaygın olarak kullanılan sıkıştırma yöntemlerinde son işlem olarak kullanılır ve muhtemelen sıkıştırma algoritmalarında en yaygın olarak kullanılan bileşendir. CCITT'nin (Consultive Committee on INTernational Telephone and Telegraph) faks iletimi için geliştirdiği 1-boyutlu kodlama, tipik bir Huffman kodlamasıdır. Huffman kodlaması verilen bir model (olasılık kümesi) için en uygun önek kodunu oluşturur.



Huffman Kodu, bilgisayar biliminde, veri sıkıştırması için kullanılan, bir entropi kodlama algoritmasıdır. David A. Huffman tarafından 1952 yılında geliştirilmiştir. Huffman'ın algoritması, her sembol (veya karakter) için özel bir kod üretir. Bu kodlar (ikilik sistemdeki 1 ve 0'lardan oluşan) bit haritası şeklindedir. Veri içerisinde en az kullanılan karakter için en uzun, en çok kullanılan karakter için ise en kısa kodu üretir. Huffman tekniği günümüzde tek başına kullanılmaz. LZW, RLE gibi yöntemlerle birlikte kullanılır.

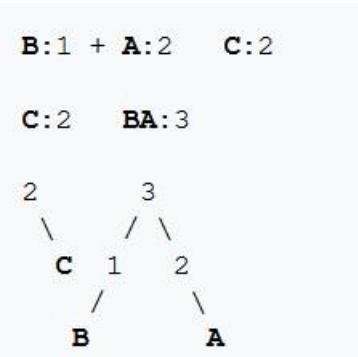
Huffman'ın algoritması, veri içerisindeki karakterlerin kullanım sıklığına (frekans) göre bir ağaç oluşturur. Ağacın en tepesinden aşağıya doğru ilerlerken sola ayrılan dal için 0, sağa ayrılan dal için 1 kodu verilir.

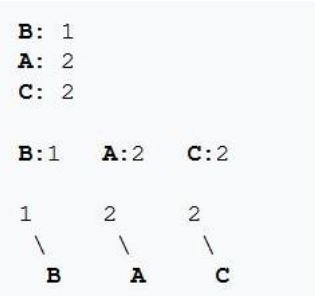
Yukarıda koyu rakamlar karakter sayısını (kullanım sıklığı-frekans) gösterir, eğik rakamlar ise bit kodlarını gösterir. Bu ağaç "ABC" karakterlerinden oluşan bir veri kümesi için üretilmiştir. Ağaca göre karakterler için bit haritaları şu şekildedir: B: 00 A: 01 C: 1 Oluşturulan bit haritaları karakterlerin veri içerisindeki konumlarına göre yerleştirilir.

Ortaya çıkan bit haritası sıkıştırılmış veridir. Örneğin; "BAACC" verisi elde edilen bit haritalarına göre yeniden düzenlenirse: 00 01 01 1 1 = 00010111 = 17h yani %80 oranında bir sıkıştırma elde edilmiş olur. İlk önce karakterlerin frekansları (kullanım sıklıkları) hesaplanmalıdır.

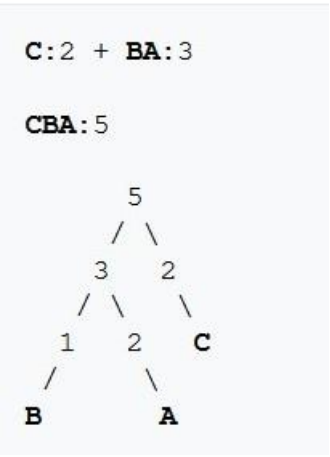
Örneğin, elimizdeki veri "BAACC" olsun,

En küçük iki frekans toplanır ve frekans tablosu yeniden düzenlenir





Tek bir ağaç oluşturulana kadar sürekli en küçük frekanslar toplanır.



❖ **DEFLATE Algoritması**

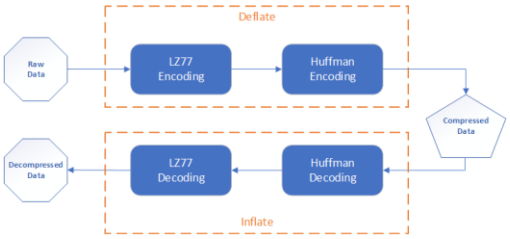
DEFLATE algoritması, Huffman ve LZSS metotlarının birleşimidir. DEFLATE algoritmasında her bir blok için önce LZSS ardından Huffman kodlaması uyguladık. Metini 64 uzunlukta bloklara bölerek sıra ile LZSS algoritmasına gönderdik. Elde edilen sıkıştırılmış metinleri de Huffman algoritmasına göndererek metinin DEFLATE ile sıkıştırılmış halini elde ettik.

DEFLATE algoritmasının üç modu var.

1. Hiç sıkıştırma yok, bloklar olduğu gibi bırakılır.
2. Her zaman sabit Huffman ağaçlarıyla mesafe ve uzunluklar yazdırılır.
3. Her bloğun Huffman ağacı var.

Bu projede DEFLATE'in 3. mod kullanılmıştır.

Blokların uzunlukları sabit ya da değişken olabilirdi. Sabit bir değer vermek istedik. Bu yüzden her



bloğun uzunluğunu 64 olarak belirledik. Eğer bir metinde 64 ten daha kısa uzunlukta bir satır var ise o satır için blok uzunluğu satırın uzunluğu kadar olur.

VI. Zorlandığımız Noktalar

Dünyanın mevcut durumu (COVID-19 krizi) yüz yüze takım olarak geliştirmemize engel oldu. Bundan dolayı başka yöntemlere başvurduk. Discord uygulamasını kullanmaya başladık ve bağlantı sıkıntıları olmasına rağmen başarılı bir şekilde projemizi geliştirdik.

Algoritmaları anlamakta zorlandık. LZ77 ve LZSS arasında bir fark olup olamadığını araştırırken çok kararsız kaldık ama LZSS algoritmasının LZ77 algoritmasının bir uyarlaması olduğu ve arasında farkların bulunduğunu öğrendik.

DEFLATE algoritmasını yazarken LZSS ve Huffman algoritmasını nasıl birleştireceğimiz konusunda zorlandık.

Görev dağılımı ise, her gün beraber Discord üzerinden konuştuğumuzdan dolayı eşittir.

VII. Sonuç

1. kaynak.txt dosyasındaki string: abracadabrarray
blok uzunluğu: 32.

LZ77sonuc.txt dosyasının içeriği:

- VERI [POINTERSIZ]: abrcdry

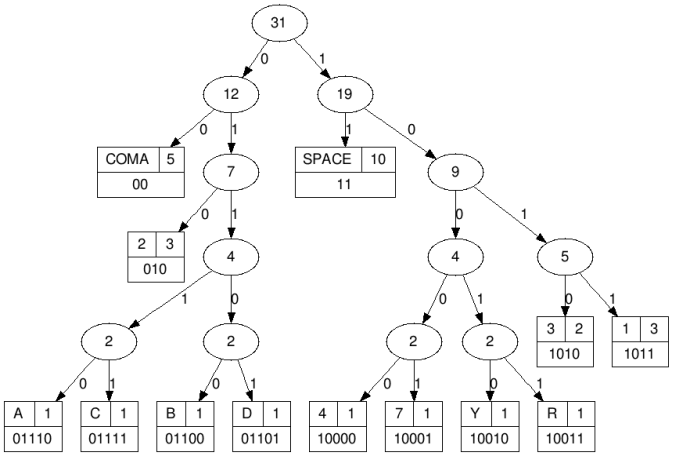
- VERI [POINTERLI]: abr <3,1,c> <5,1,d> <7,4,r> <10,2,y>

SIFRELENMEDEN ONCE BOYUT: 15 byte
SIFRLENDIKTEN SONRA BOYUT: 7 byte
1. [BULUNAN BENZETME]: <0,0,[a]>
2. [BULUNAN BENZETME]: <0,0,[b]>
3. [BULUNAN BENZETME]: <0,0,[r]>
4. [BULUNAN BENZETME]: <3,1,[c]>
5. [BULUNAN BENZETME]: <5,1,[d]>
6. [BULUNAN BENZETME]: <7,4,[r]>
7. [BULUNAN BENZETME]: <10,2,[y]>

DEFLATEsonuc.txt dosyasının içeriği:

SIFRELENECEK BLOK: abracadabrarray

LZSS'den sonra: abr 3,1 c 2,1 d 7,4 2,1 3,2 y



(dosyada görsel bulunmuyor, ağaçlanmayı gösterme amaçlıdır)

HUFFMAN CÜMLESİ: 01110011001001111101000 10111101111110100010111101101111000100 10000 11010001011111010000101110010

DEFLATE'den sonra boyut: 12 byte

2. kaynak.txt dosyasındaki metin: Ali elma yemeği çok seviyor. Ali'nin annesi ona elma veriyor.
blok uzunluğu: 32.

LZ77sonuc.txt dosyasının içeriği:

❖ VERI [POINTERSIZ]: Ali emaymycoksvyr.A'nnansoaemvryr.

- VERI [POINTERLI]: Ali e <4,1,m> a <5,1,y> <6,1,m> <8,1,y> <12,2,c> ok <16,1,s> <17,1,v> <21,1,y> <8,1,r> . <25,1,A> <29,2,'> n <20,1,n> <28,1,a> <5,1,n> <30,1,s> <28,2,o> <12,1,a> <28,1,e> <19,1,m> <5,2,v> <14,1,r> <25,1,y> <14,1,r> .

SIFRELENMEDEN ONCE BOYUT: 61 byte

SIFRLENDIKTEN SONRA BOYUT: 34 byte
(bulunan benzetmelere LZ77sonuc.txt'e bakınız)

DEFLATEsonuc.txt dosyasının içeriği:

SIFRELENECEK BLOK: Ali elma yemeyi cok seviyor. Al

LZSS'den sonra: Ali e 4,1 ma 5,1 y 6,1 5,1 2,1 4,1 12,2 cok 4,1 s 9,1 v 9,1 11,1 8,1 r. 9,1 29,2

HUFFMAN CUMLESI:

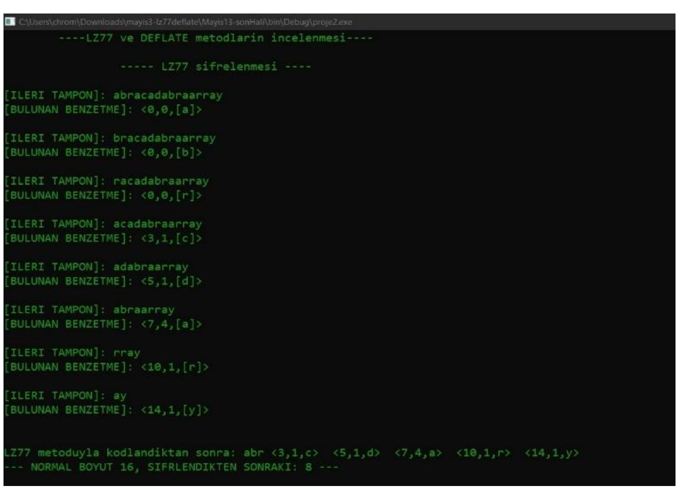
10010010110101001011111001010110111110100110110110100001101001101001101011111011000101001111010011010011111000101001111011111010011110010001011000110100011001000010101110111110100110101001110011101001101110111100111010011110000101001111011001101001101110001011011100111010011111100010011101100011

SIFRELENECEK BLOK: i'nin annesi ona elma veriyor.

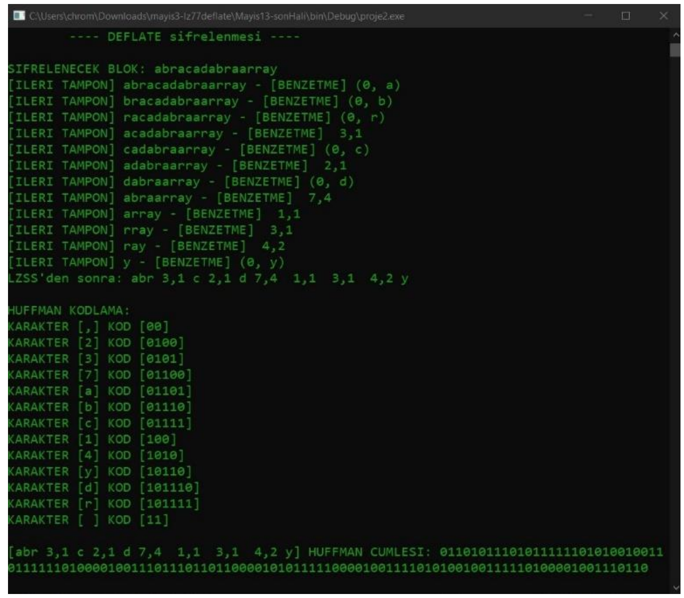
LZSS'den sonra: i'n 3,1 2,1 a 3,1 1,1 es 8,1 7,1 o 6,1 9,1 4,1 8,1 lm 5,2 v 6,1 r 14,1 y 14,1 4,1 .

HUFFMAN CUMLESI:
0001101001010000100111001111010111111001101010111110010001110011110101111101101011100100110010011110000101011111000101101011110010001110001101011111001110101011111000010101111110000101011100011100110111001011101100110110010101110001101011100110011010000101011100111111010000101011111000010101111001011

DEFLATE'den sonra boyut: 74 byte
Giriş ekranında kodlamanın tüm adımları gösterilmiştir. Aşağıda LZ77'nin adımları gösterilmiştir.

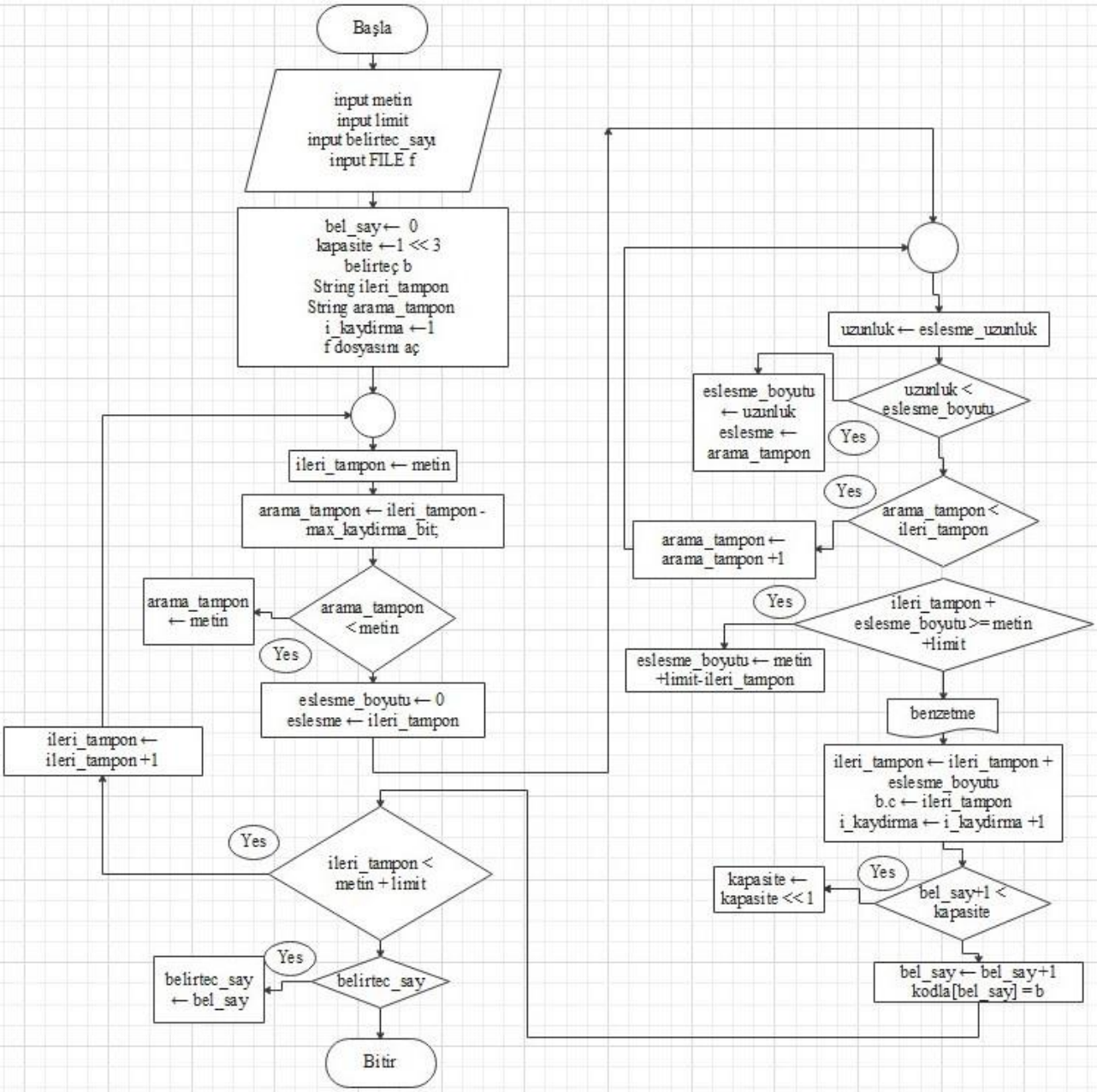


DEFLATE algoritmasının LZSS ve Huffman kodlama adımları aşağıda gösterilmiştir.



Son olarak, hangi algoritmanın daha iyi sıkıştırma yaptığını yazdırılır.

LZ77 Akış Diyagramı



VIII. KAYNAKÇA

1. https://ysar.net/algoritma/lz77.html
   (Erişim Tarihi: 10.04.2020)

2. https://www.slideshare.net/veysiertekin/lz77-lempelziv-algorithm
   (Erişim Tarihi: 11.04.2020)

3. https://en.wikipedia.org/wiki/LZ77_and_LZ78
   (Erişim Tarihi: 18.04.2020)

4. https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Storer%E2%80%93Szymanski
   (Erişim Tarihi: 19.04.2020)

5. https://en.wikipedia.org/wiki/Huffman_coding
   (Erişim Tarihi: 23.04.2020)

6. https://en.wikipedia.org/wiki/DEFLATE
   (Erişim Tarihi: 25.04.2020)

7. https://ramazanakbuz.com/lz77-lz88-sikistirma-algoritmasi/
   (Erişim Tarihi: 12.04.2020)

8. https://zlib.net/feldspar.html
   (Erişim Tarihi: 10.04.2020)

9. https://tools.ietf.org/html/rfc1951
   (Erişim Tarihi: 16.04.2020)

10. https://stackoverflow.com/questions/25691746/DEFLATE-algorithm-pseudocode
    (Erişim Tarihi: 23.04.2020)

11. http://altanmesut.trakya.edu.tr/pubs/dr_tez.pdf
    (Erişim Tarihi: 08.04.2020)

12. https://tr.qwe.wiki/wiki/LZ77_and_LZ78
    (Erişim Tarihi: 14.04.2020)

13. https://openzipfile.net/lz77-DEFLATE.html
    (Erişim Tarihi: 20.04.2020)