VERİ SIKIŞTIRMA PROJESİ

Kocaeli Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği

Taha Yıldız
tahayildiz515@gmail.com
180201019

Ahmet Furkan Kaşıkçı
furkankasikci45@gmail.com
180201030

ÖZET

Bu projede amaç **“LZ77”** algoritmasını ve **“Deflate”** algoritmasını kullanarak veri sıkıştırmak ve bu algoritmaları karşılaştırmaktır.

Anahtar Kelimeler:

Deflate,lz77,veri,sıkıştırma.

1.Giriş

1.1 LZ77 Algoritması:

“LZ77 Algoritması” verideki tekrar eden bölümleri ortadan kaldırıp sıkıştırmayı sağlar. **“kaynak.txt”** ‘deki **“abracadabrarray”** ‘den yola çıkılırsa ilk olarak karakterlerin ne kadar sayıda tekrar ettiğine bakılır. Daha sonra karakterlerin verideki yerlerine bakılır. Karakterlerin yerlerine **“offset”**, karakterlerin sayısına **“lenght”** denir. Bunlar ilki **“offset”** olmak üzere **“<”** ve **“>”** işaretleri arasında gösterilir. Bu gösterime ise **“token”** denir. Kodda tokenler **“token”** structında tutulur. **“char”** kısmı verideki karakteri tutarken **“offset_len”** ise **“token”** işaretini tutar . Her bir token beş **“offset”** biti üç **“lenght”** biti ve sekiz **“karakter”** biti olmak üzere toplam onaltı bit(iki byte) genişliğindedir. Bu sebeple **“offset”** maksimum otuzbir, **“lenght”** maksimum yedi bit olabilir. Bunlar koddaki makrolar kısmında görülebilir.

Tokenleri oluşturmak için kullanılan **“*LZ77”** fonksionuna gelindiğinde,bu fonksiyon struct token tipindedir. Parametreleri ,sıkıştırılacak metni alan **“*metin”**, sıkıştırılacak karakter sayısını tutan **“boyut”** ve kullanılan token sayısını gösteren pointer olan **“*tokenSayisi”** dir. Fonksiyon içindeki değişkenler ise kaç tokenlik yer ayrıldığını tutan **“yer”** , oluşturulan token sayısını tutan ve başlangıç değeri sıfır olan **“_tokenSayisi” ,** oluşturulacak tokeni tutacak olan token tipinde **“t”, “*sifrele”** token dizisi ve bunun için yer değişkeninin değerine göre yer ayırılır**,** tokenleri oluştururken metni iki parçaya böleceğimiz için kullanacağımız **“*ileri”** ve **“*ara”** değişkenleri de burada tanımlanır. Metnin bir tarafına **“*ileri”** bir tarafına **“*ara”** gelecek ve **“*ara”** içinde **“*ileri”** nin eşleşmesi aranacak kod bu eşleşmeyi bulduktan sonra tokenin **“offset”**, **“lenght”** değerlerini de belirleyecek,bu değerler de belirlendikten sonra **“*ara” , “*ileri”** ileriye doğru kaydırılacak. Kod **“*ara” ‘**nın karakter genişliği otuzbir karakter, **“*ileri” ‘**nin karakter genişliği ise yedi karakter olarak belirledi. Bundan sonra kod token oluşturma döngüsüne girer. Döngü içine gelindiğinde metni ikiye bölüp bir tarafa **“*ara”,**bir tarafa **“*ileri”** konduğu için ve **“*ara” ‘**nın karakter genişliği otuzbir karakter olduğu için **“*ara” ‘**yı ilerinin otuzbir karakter gerisine koyuyoruz.Daha sonra **“*ara”** ve **“metin”** karşılaştırılır.

“metin” “*ara” ‘dan büyükse **“*ara” “metin”** ‘eşitlenir ve böylece **“*ara”** ‘nın metnin dışına çıkmasına engel olunur. Başlangıç değeri sıfır olan **“*ara”** ‘nın bulduğu en büyük eşlemenin boyutunu tutacak olan **“max_len”** tanımlanır. Bu eşleşmenin pozisyonunu tutacak char tipinde olan **“*max_match”** tanımlanır. Sonrasında kod bu döngü içinde **“*ara”** içinde arama yapmak için bi döngüye daha girer.Burada tanımladığı integer tipindeki **“len”** değişkenini **“eslesenUzunluk”** fonksiyonuna eşitler.Bu fonksiyona **“*ara”, ”*ileri”** ve en fazla nereye kadar kontrol yapılacağını gönderir.Bu fonksiyonda gönderilen **“*ara”** ve **“*ileri”** ‘nin ilk kaç karakteri özdeş gönderilen **“boyut”** dahilinde ona bakılır ve ilk kaç karakter özdeşse onun sayısı döndürülür. Döngü içine dönüldüğünde **“len”** değişkeni **“max_len”** değişkeninden büyükse burada değişim işlemleri gerçekleşir.Bu döngüden çıkıldıktan sonra eğer eşleşmeye metnin son karakteri de dahilse bir karşılaştırma yapılır.Eğer **“ileri”** ve **“max_len”** ‘in toplamı büyük eşit **“metin”** ve **“boyut”** ‘un toplamıysa **“max_len”** ‘inin boyutu değiştirilir. Bunun nedeni tokenin içine bir karakter daha koyabilmek için maksimum eşleşmeyi kısaltmaktır.Daha sonra başta tanımlanan **“t”** tokeni bulunan eşleşmeye göre oluşturulur ,**“ileri” “max_len”** kadar artırılır ve **“t”** ‘nin **“c”** değişkenine eşitlenir. Bundan sonra eğer **“_tokenSayisi” ‘**nin bir fazlası başta ayırılan **“yer”** ‘den büyükse **“sifrele”** ‘nin boyutu güncellenir.En sonunda bulunan eşleşmeye göre oluşturulan **“t”** tokeni **“sifrele”** ‘ye kaydedilerek bir döngü sonlanır.Döngüden sonra tokenler yazdırılır ve **“sifrele”** döndürülür.

“lz77_Main” ise **“*okunacakDosya”** adlı bir parametresi vardır.Önce bu dosya açılıp **“file_read”** fonksiyonuyla karakter karakter okunur.Daha sonra tanımlanan **“sifrele_metin”** tokeni **“*kaynak_metin”** , **“metin_boyutu”** , **“token_sayisi” ’**ni parametrem olarak alan **“LZ77”** fonksiyonuna eşitlenir ve buradan gelecek token işaret dizisi **“şifrele_metin”** ‘e eşitlenmiş olur.Daha sonra bu token işaret dizisi **“ekranaYaz”** fonksiyonuyla ekrana yazdırılır.Eğer okunan dosya **“kaynak.txt”** dosyası ise dosyaya yazma işlemi **“lz77.txt”** dosyasına yapılır.Eğer **“trash.txt”** ise okunan dosya **“deflate.txt**” ‘ye yazma işlemi gerçekleşir.

LZ77 Kaba Kod:

-Kaynak metni ikiye böl ,iki tarafa pointer yerleştir ve **“LZ77”** fonksiyonu sayesinde eşleşme ara.

-Tokenin değerlerini belirle ve pointerları kaydır.

-En büyük eşleşmeyi bul ve token işaret dizisi döndür.

- Sonrasında **“lz77_main”** ‘de okunan dosya ismine göre **“lz77.txt”** veya **“deflate.txt”** dosyalarına yaz.

1.2 Huffman Algoritması:

Öncellikle **“node”** structı tanımlanır. Bu struct sayesinde bir Huffman Ağacı düğümü oluşturulur.Bu structın içinde bir karakter ,bu karakterin değeri ve bu düğümün çocuk düğümleri tutulur. **“alfabeTekrar”** dizisi tanımlandı. Bu dizide İngiliz alfabesinde bulunan her bir karakterin değeri bir nevi frekansı tutulur. **“kucukBul”** fonksiyonuna gelindiğinde bu fonksiyon Huffman ağacındaki en küçük düğümü bulur ve döndürür.Bu fonksiyonda başlangıç değeri sıfır olan bir **“i”** değişkeni ve gene integer olan **“kucuk”** değişkeni vardır vardır.Parametre olarak alınan düğüm dizisinde bir döngüyle “i” kullanılarak değer değişkeni eksi bir olan indis **“kucuk”** ‘e eşitlenir.Sonrasında **“naber”** de **“i”** ‘ye eşitse tekrardan “değer” ‘i eksi bire eşit olan dizi elemanını bulmaya çalışılır. Bulunduğunda yeni **“i” “kucuk”** ‘e eşitlenir. Sonrasındaki her karakter için eğer o karakterin **“deger”** ‘i eksi bire eşit değilse veya indisi **“naber”** ‘e eşit değilse bir karşılaştırma yapılır.Eğer o indisin **“deger”** ‘i bir önceki döngüde bulunan **“kucuk”** ‘un **“deger”** ‘inden küçükse yeni düğümün en küçüğü bulunmuş olur döngü sonunda bulunan en küçük döndürülür. **“huffmanAgaci”** fonksiyonunda ise hufmann ağacı oluşturulur ve adresi döndürülür.Önce düğümün her elemanı için bellekte yer ayrılır.Daha sonra **“alfabeTekrar”** ‘a göre değerleri girilir. **“sol”** ve **“sag”** elemanları boştur. Sonraki döngüde ise **“kucukBul”** fonksiyonu yardımıyla sıralanır ,her düğüm elemanın **“sag”** ve **“sol”** ‘ları da böylelikle doldurulmuş olur.Parametre olarak alınan elemana ise oluşturulan ağacın küçüğü atılır.

“kodlaDoldur” ise her karakter için bit tablosu oluşturur. **“sifrele”** fonksiyonu ise girilen dosyayı sıkıştırmaya yarar. Girilen dosyanın **“fgetc()”** ile dönen değeri ona eşit olmayıncaya dek bir döngü çalışır.Bu değer **“c”** değişkenine eşitlenir.Eğer bu değer otuzikiye eşit ise parametre olarak alınan kodla dizisinin son elemanı **“len”** makrosuna gönderilir ve fonksiyon başında tanımlanan **“uzunluk”** değişkenine eşitlenir.Gene tanımlanan **“n”** değişkeni **“kodla”** ‘nın son elemanına eşitlenir.Eğer bu **“if”** ‘e girmesse son indis yerine **“c-97”.** İndis kullanılır.Bu sorguların birinden gelen uzunluk sıfıra eşit olasıya kadar bir döngüye daha girilir.Burada bu uzunluk her bir döngüde azaltılır. **“n”** ise her seferinde ona bölünür ve ona modu alınıp bir çıkartılır ve “bit” değişkenine eşitlenir. Başlangıç değeri sekiz olan **“bitSol”** değişkeni ise her seferinde azaltılır ve sıfıra gelince tekrardan sekiz yapılır ve başlangıçta tanımlanan **“x”** değişkeni **“fputc()”** ‘ye **“output”** ile birlikte gönderilir ve sonrasında sıfıra eşitlenir.En son bu döngülerden çıktıktan sonra **“bitSol”** sekize eşit değilse **“x” “bitSol-1”** değerine göre sola kaydırılır ve ve gene **“fputc()”** ‘ye **“output”** ile birlikte gönderilir. **“degistir”** fonksiyonu parametre olarak alınan **“kodla2”** dizisini ters çevirerek **“sifrele”** fonksiyonu yardımıyla mod operatörü ile kullanılmasını sağlar. Gene parametre olarak alınan **“kodla”** dizisinin her elemanı yedek değişkene kaydedilir ve yanında **“kopya”** değişkeni tanımlanır.Bu kopya değişkeni onla çarpılır ve yedek değişkenin ona modu alınır ve gene toplamı **“kopya”** ‘ya kaydedilir.En sonunda yedek değişken sıfırdan küçük olunca **“kopya” “kodla2”** ‘ye kaydedilir. **“huffman_Main”** ‘de ise dosya açma kapama ve değişkenleri fonksiyonlara gönderme işlemleri yapılır.

Huffman Kaba Kod:

-İlk olarak huffman ağacı oluştur.

-Her karakter için bit tablosu oluştur (kodlaDoldur).Burada **“+1”** 0, **“+2”** 1 anlamına gelir .

-**“degistir”** fonksiyonunu kullan.**“degistir”** fonksiyonu parametre olarak alınan “kodla2” dizisini ters çevirerek **“sifrele”** fonksiyonu yardımıyla mod operatörü ile kullanılmasını sağlar.

-**“sifrele”** kullan.Bu fonksiyon girilen dosyayı sıkıştırmaya yarar.

-En son ise huffman ağacı oluşturulmuş ve skıştırılmış olup **“lz77”** ‘ye gönder ve böylece Deflate uygulanmış olur.

1.3 Deflate Algoritması:

Huffman ve LZ77’nin birleştirilmesi oluşan algoritmadır.Kodda ise bu Huffman Ağacının oluşturulup ve sıkıştırılıp ve tekrar sıkıştırılması için LZ77 ‘ye gönderilmesiyle olur. Sıkıştırılacak olan veri birbirini takip eden bloklar kümesi olarak düşünülür.Her blok için oluşturulan Huffman ağacı bir önceki ve bir sonraki bloktan bağımsızdır.Sıkıştırılabilen blokların büyüklüğü değişkendir.Deflate algoritması Huffman ağacının etkili kodlama yapamayacak kadar büyüdüğünü gördüğünde, yeni bir Huffman ağacı oluşturmak için o bloğu sonlandırarak yeni bir blok başlatır.Sıkıştırılamayan blokların boyu ise byte ile sınırlıdır.

2.Karşılaşılan Sorunlar

-Deflate ve Huffman algoritmaları hakkında yeterli kaynak bulunamaması.

3.Çalışma Adımları

1)

```
lz77 icin 1, deflate icin 2, programi sonlandirmak icin 3 girin.
1
<0,0,a>
<0,0,b>
<0,0,r>
<3,1,c>
<5,1,d>
<7,4,r>
<10,2,y>
<0,0,
>
```

2)

```
lz77 icin 1, deflate icin 2, programi sonlandirmak icin 3 girin.
2
```

3)

```
lz77 icin 1, deflate icin 2, programi sonlandirmak icin 3 girin.
3
lz77 ile sifrelenen dosyanin boyutu: 16
 deflate ile sifrelenen dosyanin boyutu: 14
```

Kaynak Dosya Metni

*kaynak.txt - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

abracadabrarray

LZ77 ile sıkıştırılmış metin

*lz77.txt - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

a b r□c)d<rRy

Deflate ile sıkıştırılmış metin

*deflate.txt - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

6 ô ş d"÷ L Ø

4.Kaynakça

1)
https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/

2)
http://bilgisayarkavramlari.sadievrenseker.com/2009/02/25/huffman-kodlamasi-huffman-encoding/

3)

https://en.wikipedia.org/wiki/DEFLATE

4)

https://www.quora.com/How-does-the-DEFLATE-compression-algorithm-work

5)
https://www.slideshare.net/veysiertekin/lz77-lempelziv-algorithm