Algoritmalar ile Dosya Sıkıştırma Projesi

Enes TELLİ

Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi

180201099

enes1903telli@gmail.com

Serhat ÇUBUKÇUOĞLU

Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi

180201032

serhatcubukcu@outlook.com

# Özet

Bu projenin amacı,kullanıcının verdiği “metin.txt” dosyasının içeriğini LZ77 algoritması ve DEFLATE algoritması kullanarak iki farklı biçimde sıkıştırılmasını sağlamak.Sonucunda ise bu iki farklı sıkıştırılmış dosyayı boyutsal olarak karşılaştırarak hangisinin verimli olduğunun gösterilmesi.

Proje C dili kullanılarak geliştirilmiştir.Program çalıştırılırken dosyanın içeriği tutulmuş sonrasında bu içeriğin işlenmesi ve dönüştürülmesi için önceden tanımlanmış makrolar,sabit değişkenler,değişkenler,struct yapılarıyla düğümler,fonksiyonlar kullanılmıştır.

**1.Giriş**

Bu projede,verilen dosyanın içeriğinin bir fonksiyon ile okunup sonrasında kullanılabilir halde tutulması gelecek işlemlerin ilk adımıdır.

Devam adımlarının bir kolu olarak LZ77 algoritmasını gerçekleştirmemizi sağlayacak uygun struct yapısının oluşturulması.Diğer bir kolu olarak da Huffman kodlamasının uygun struct yapısı ile oluşturulması ve sonrasında bu kodlamadan elde edilen bilgilerin sıkıştırılması ile DEFLATE algoritması oluşturulması.

Bu temel yapı hazırlandıktan sonra sıkıştırılan dosyalar boyutsal olarak karşılaştırılabilir ve kullanıcıya karşılaştırma sonucu sunulabilir.

**2.Yöntem**

Öncelikle “metin.txt” dosyası “file\_read” fonksiyonu ile okutulur.Dosyanın içeriği “kaynak\_metin”nde tutulurken,dosyanın boyutu da “metin\_boyutu”nda tutulur.Tanımlanan “token” structı ile “offset\_len” ve “c” değişkenleri olan sabit iki bytelık tokenler oluşturulması amaçlandırılmıştır.Bu doğrultuda tokenleri oluşturup sıkıştırılmış metin haline getirecek “struct token \*encode” fonksiyonu tanımlanır.Bu fonksiyon ana fonksiyondan “kaynak\_metin”,”metin\_boyutu”,”token\_sayisi” ve kontrol vazifesi gören “k” değişkenini alır.Bu fonksiyonda hafızadaki token sayısını ayarlayan “cap”,oluşturulacak token olan “t”,token sayısını ana fonksiyona taşımaya yarayacak “\_NumTokens”,arama ve ileri tampon görevi görecek olan sırasıyla “search” ve “lookahead”,oluşturulan tokenleri bir diziye kaydetmemizi sağlayacak “encoded”,en uzun eşleşmeyi bulmamızı sağlayacak “max\_match”,en uzun eşleşmenin boyutunu tutan “max\_len” ve boyut tutan “len” tanımlanır.Sonrasında bir for döngüsü ile ileri tamponda tüm metin boyunca aramaya başlarız.Her bir tekrarda arama tamponu, ileri tamponun “#define” olarak tanımladığımız OFFSETMASK(x)’tan çıkarılması değerini alır.Bu sayede arama tamponunun boyutu belirlenir.

Arama tamponu içerisinde arama yapmak için ise ayrıca bir döngü kullanılır.Kullanılan bu döngüde “prefix\_match\_length” fonksiyonu iş görür.Bu fonksiyona yollanan karakterler while döngüsü içerisinde biri arama tamponundaki karakter diğeri ileri tampondaki karakter olmak üzere eşleşe eşleşe ilerlerler.Eğer eşleşme durumu bitti ise o zamana kadar gerçekleşen eşleşme sayısını döndürülür.Bir if komutu ile bu eşleşme sayısı,en yüksek olanı ise “max\_len” bu değeri alır ayrıca “max\_match” arama tamponunun karakterini tutar.

Bu bilgiler elde edildikten sonra token oluşumu gerçekleşir.”t.offset\_len” değeri “#define” olarak tanımlanan OFFSETLENGTHMASK(x,y)’ye sırasıyla “lookahead” ve “max\_match” yollanır,sonuç olarak tokenin offseti elde edilir.Tokenin en son eşleşmesinden bir sonraki değerini elde etmek için ise “t.c” değeri “lookahead”e eşitlenir bu sayede token tamamlanır.

Tamamlanan bu tokenler bir dizide tutulmalıdır ki bir sıraya sahip olunabilsin ve sıkıştırma yapılmış olsun.Bunu “struct token” olarak oluşturulmuş “encoded” dizisi halleder.Son,olarak token sayısı en son halini alır ve “encoded” dizisi ana fonksiyona döndürülür encode\_metin eşitliğini sağlar.Bu metin,oluşturulan token sayısına bağlı olarak LZ77 algoritması ile sıkıştırılmış dosya “LZ77”yi oluşturur.Aynı zamanda sıkıştırılmış dosyanın boyutu “lz77” değişkeninde karşılaştırma yapmak için tutulur.

Huffman kodlamasını oluşturmak için

**Sözde kod:**

Komsu struct yapısı tanımlanır;

Sehir struct yapısı tanımlanır;

int main()

{

İlk düğüm belirlenir;

Bağlamayı sağlayacak düğüm belirlenir;

Dosya okunmak için açılır;

while(Açılan dosya satır satır okunur NULL’a kadar)

{

plaka kodu,sehir adı,coğrafi bölge ve kaç eleman olduğu bilgisi tutulur;

}

Dosya başına gelinir;

while(Açılan dosya satır satır okunur NULL’a kadar)

{

Burada satır satır okunan bilgi düğümlere aktarılır;

}

for()

{

En büyük komşu sayısı belirlenir;

}

while(1)

{

Menü yapısı yazdırılır;

Secim işlemi kullanıcıya tanımlanır;

switch(secim değerine göre koşul seçilir)

{

case 1;

Kullanıcıdan komsu ya da şehir eklemek için tercih alınır;

if(komsu tercihi)

{

Hangi şehre hangi komşuluğu ekleyeceği alınır;

Eklenecek şehir mevcutsa işleme devam eder;

komsu\_ekle() fonksiyonu ile her iki şehre komşuluk tanımlar;

}

if(şehir tercihi)

{

Eklenecek şehrin bilgilerini alır;

Alınan bilgiler bir başka şehirle çakışmıyorsa işleme devam eder;

Alınan bilgiye göre plaka kodu büyüklüğü ne ise ona göre başa ortaya ya da sona eklemeyi fonksiyonlarla gerçekleştirir;

}

break;

case 2;

Alınan plaka kodu ya da şehir adı bilgisine göre aranan şehri düğüm içerisinde tarayarak ekrana bilgilerini yazdırır;

break;

case 3;

Kullanıcıdan komsu ya da şehir silmek için tercih alınır;

if(komsu tercihi)

{

Hangi şehrin komşu kaydının silineceğini alır;

Eğer o şehir mevcut ise silinecek komşu kaydının bilgisini alır;

Silinecek komşu kaydı da mevcut ise remove\_komsu fonksiyonuna yollanarak karşılıklı olarak komşuluklar silinir;

}

if(şehir tercihi)

{

Silinecek şehir bilgisi alınır;

Eğer şehir mevcut ise öncelikle onun komşuluklarını bir diziye alıp remove\_komsu fonksiyonuna yollayıp komşulukları kaldırır;

Şehir bilgisini remove\_by\_index fonksiyonuna yollayarak şehri düğümden siler;

}

break;

case 4;

Kullanıcıdan bölge bilgisi alır;

for()

{

Alınan bilgiye uygun şehir bulursa yazdırır;

}

break;

case 5;

Kullanıcıdan komşu sayısı kriteri alır;

Kriterin tanımına göre if komutlarında o komşu sayısına sahip şehirler bulunur ve ekrana yazdırılır;

break;

case 6;

Kullanıcıdan alt üst komşuluk sayısı değerleri alınır;

Ortak komşu olacak şehir adları alınır;

while(alt komşuluk sayısından üst komşuluk sayısına kadar)

{

Ortak komşu stringine sahip olan şehirleri yazdırır;

}

break;

case 7;

plaka\_arit() fonksiyonu ile plakalar komsu plakaları sıralanır;

print\_list() fonksiyonu ile cikti.txt dosyasına düğüm yapısı yazdırılır;

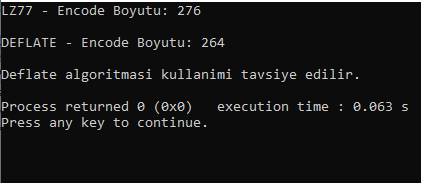
break;

}

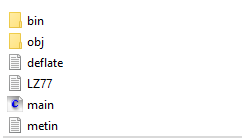
}

**3.Deneysel sonuçlar**

1)Konsol



2)Girdi ve Çıktılar



**4.Sonuçlar**

Proje,klasör içerisindeki dosya içeriğini elde edip bu bilgileri hem LZ77 hem DEFLATE algoritmaları için sıkıştırabilir.Sıkıştırılmış bu dosyaları boyutuna göre karşılaştırabilir ve kullanıcıya hangisinin seçilmesi gerektiğini belirtebilir.

**5.Kaynakça**

[1] “LZ77 algoritması”

https://ysar.net/algoritma/lz77.html

[2] “Seçerek sıralama”

<http://bilgisayarkavramlari.sadievrenseker.com/2008/08/09/secerek-siralama-selection-sort/>

[3] “Huffman Kodlaması”

https://ysar.net/algoritma/huffman-kodlamasi.html

[4] “C Operatörleri”

<https://www.tutorialspoint.com/cprogramming/c_operators.htm>

[5] “Önişlemci Direktifleri ve Makrolar”

<https://www.bilgigunlugum.net/prog/cprog/2c_onislemci>

<https://books.google.com.tr/books?id=3WtuCwAAQBAJ&pg=PA134&lpg=PA134&dq=önceden+tanımlanmış+makrolar&source=bl&ots=vm4cbSp4XK&sig=ACfU3U1_-hFRR2BAGeoGxaMFGSwGmSAMpw&hl=tr&sa=X&ved=2ahUKEwjZzsL4_brpAhWP3oUKHdK9AUwQ6AEwA3oECAgQAQ#v=onepage&q=önceden%20tanımlanmış%20makrolar&f=false>

[6] “Sabit Değişkenler”

<http://ozenozkaya.com/blog/?p=186>