DEFLATE ALGORITMASI

Sefa Mert GÜNGÖR
Bilgisayar Mühendisliği
180201086
KOCAELİ ÜNİVERSİTESİ
meertgngr5255@hotmail.com

Miraç Onur SAKA
Bilgisayar Mühendisliği
180201081
KOCAELİ ÜNİVERSİTESİ
m.onur7@hotmail.com

Özetçe- Bu projemizde lz77 ve huffman algoritmasını kullanarak deflate algoritmasını elde etmeye çalıştık. Bu projemizde 2 algoritmanın birbirleriyle çalışmalarını sağlamak amacımızdı. Ancak çok fazla araştırma yapmamıza rağmen deflate hakkında yeterli bilgi düzeyine ulaşamadık bu nedenden dolayı deflate için bir algoritma yazamasak da deflate için gerekli olan huffman ve lz77 algoritmalarını ayrı ayrı yazarak 2 farklı şekilde aynı dosyaları sıkıştırmış olduk.

Anahtar kelimeler- lz77, huffman, deflate, token, dizi

I.GİRİŞ

Proje klasörümüzün içinde bulunan "deneme.txt" dosyasından aldığımız yazıyı dizimize attık ardından lz77 ve huffman sıkıştırma algoritmalarını uygulamaya sırasıyla koyduk. Proje için kullanılmaya uygun görülen dil C 'yi kullandık.

II. HAZIRLIKLAR VE BILGILER

Bu proje C programlama dilinde geliştirilmiş olup, geliştirme ortamı olarak "Code Blocks 13.12" kullanılmıştır. İlk önce bizden istenen durumları değerlendirip tartışarak ne yapmamız gerektiği hakkında karara varıp projeyi hangi adımlara göre takip edeceğimize karar verdik. Daha sonrasında projeye başladık.

III. YÖNTEM

İlk olarak "deneme.txt" den istediğimiz bilgileri okuduk. Daha sonra okuduğumuz bilgileri lz77 algoritmasını attık. Lz77 algoritması gerekli işlemleri yaparak tokenleri oluşturduk ve kullanıcının karşısına konsol ekranında çıkartmış olduk. Ayrıca "lz77.txt" dosyasına tokenleri yazdırdık. Aynı şekilde "deneme.txt" den istediğimiz bilgileri okuduktan sonra okduğumuz bilgileri huffman algoritmasına attık. Huffman algoritması bilgiler üzerinde gerekli işlemleri yaparak sıkıştırmayı tamamlamış oldu ve bu şekilde sıkıştırılan verileri konsol ekranına yazdırdık.

Projede kullanılan fonksiyonlar:

"char *dosya_okuma()" Dosyadaki bilgileri programdaki diziye aktarmayı sağlar.

"int esitlik_uzunlugu()" Karakter tekrar sayısını döndüren fonksiyon.

“void encode()” Tokenleri oluşturan fonksiyon.

“void tokensay()” Token sayısını döndüren fonksiyon

“void yer_ayarla()” Frekansı en yüksekten en düşüğe doğru sıralar.

“struct dugum *yeniDugum()” Yeni harf düğümü oluşturur ve bu düğümde frekans bilgisi de tutulur.

“struct agac *nt()” Ağacın alan ve boyut bilgilerini ekleyerek ağacı yapılandırır.

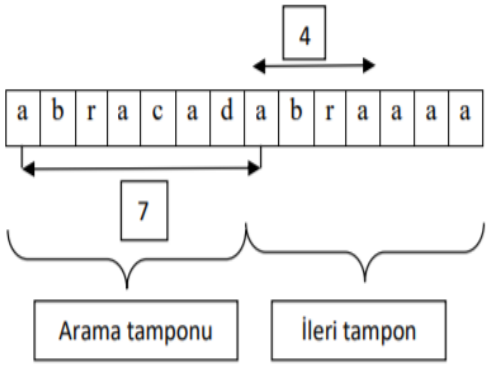
“void *dugumEkle()” Düğümü ağaçta gerekli yere ekler.

“struct dugum *dugumSil()” Ağaçtaki en küçük düğümü siler.

“void ayarla()” Ağacın şekline göre ikili numaralandırma, frekans değerlerini gösterme işlemlerini yapar.

LZ77 ALGORİTMASI

Öncelikle metin üzerinde kaç karakter geri geldiğimizi ,tekrar eden karakter sayısını ve harf bilgisini tutan bir struct yapısı tanımlamış olduk. Tanımladığımız bu yapı tokenlerimizin offset, length ve karakter değerlerini tek tek tutmuş oldu .Böylelikle tokenlerimiz bu yapı sayesinde tutabilmiş olduk. Ardından buyuk_metin adında bir değişken tanımladık daha sonra proje dosyamızın içindeki metin dosyamızın içindeki bilgileri dosya_okuma fonksiyonu aracılığıyla okuduk ve buyuk_metin değişkenimizin içine attık. Metin okuma işleminden sonra bu oluşturduğumuz buyuk_metin değişkenimizdeki metinimizi encode fonksiyonunun içine attık daha sonrasında lookhead ve search pointerları yardımıyla cümle üzerinde gezinerek tokenlerimizi oluşturduk. Burada lookhead ileri tamponu tutan, search ise geri tapmonu yani arama yapabileceğimiz yerin adresini tutan pointer olmuş oldu.



Buradaki metinimizi lz77 algoritmasına göre inceleyecek olursak. Burada ileri tamponumuzun ilk elemanı olan “a”karakteri geri tamponda yani arama tamponunda baştan ileri tampona kadar arama yapcaktır. Bizim projemizde arama tamponun ilk elemanını adresini tutan search pointerı mevcuttur. Bu search pointerı ileri tamponun arama yapılacak karakterinin adresinin 255 karakter gerisinden başlar arama tamponunun genişliğini böylelikle belirlemiş olduk. Arama tamponunu 255 karakter geriden başlatmamızın sebebi 8 bitlik bir değişken kullanacağımızdan. Daha sonra ileri tampondaki karakterimizi geri tamponda bulduğumuz değerin kaç karakter geride olduğunu, kaç karakter tekrar ettiğini 8 bitlik bir değişkenin içine atıyoruz fakat hala geri tamponda arama yapmaya devam ediyoruz eğer tekrar eden karakter sayısı fazlaolan bir blok görürsek bu sefer 8 bitlik değiikende tuttuğumuz offset ve length değerini bulmuş oluyoruz. Daha sonra ileri tamponumuzu length değeri kadar ilerletiyoruz ve bir sonraki karakteri structımızda tanımladığımız karakter değişkenimiz ile tokenimiz için tutmuş oluyoruz. Bu işlemleri yaptığımızda yukarıdaki metinimizde a için [7,4,a] şeklinde bir token oluşturmuş oluyoruz.

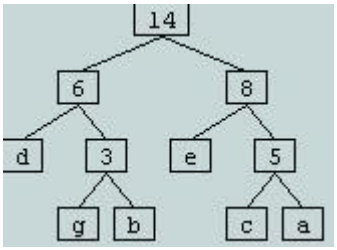
İleri tamponun sonuna geldiğimizde lz77 ile metinimizi sıkıştırmış olduk

Bu şekilde lz77 algoritması ile oluşturduğumuz tokenlerimizi ana fonksiyonda oluşturduğumuz lz77.txt dosyamızın içine bastırıyoruz. Böylelikle lz77 algoritmamızın sonuna gelmiş oluyoruz.

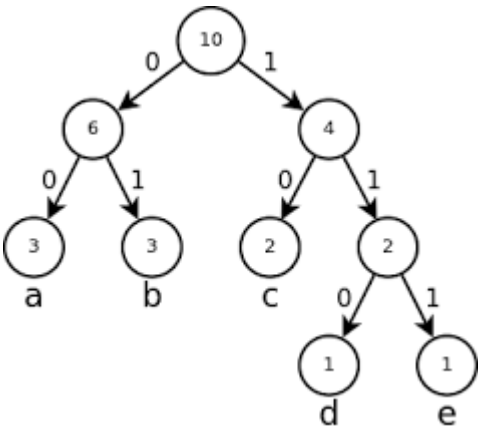
HUFFMAN ALGORİTMASI

Bir metinde tekrar eden harfleri sayısına göre sıraya yerleştirip onları temsili ikili kodlarıyla veren algoritmadır.

Öncelikle "deneme.txt" den bilgileri okuyup bir diziye attık. Dizide bütün metindeki harflerin tekrar sayısını bulup her harfin tekrar sayısını frekans isimle diziye sıra sıra yerleştirdik. Daha sonra algoritma gereği en büyük frekans sayılı harfi en başa koyduk. Çünkü bu şekilde en çok tekrar eden harf en az ikilik kodlarla temsil edileceği için hafızada daha az yer kaplayacaktır.



Resimdeki örnek gibi şematize edilebilir.



İkili numaralandırma ise şekilde görüldüğü gibi ağacın sağa doğru giden dallarına 1 sola doğru giden dallarına 0 verilir.

En sonda yerleştirilen harfler temsili gösterilmesi için doğru fonksiyona giderek işlem tamamlanır.

IV. YALANCI KOD

1-Program çalıştırıldı.

2-"deneme.txt" den bilgiler okundu.

3-Okunan bilgiler diziye atıldı.

4-Dizideki bilgiler lz77 algoritmasına gönderildi.

5-Lz77 algoritması sonucunda tokenler oluştu.

6-Oluşan tokenler ve sıkıştırılmış dosya boyutu konsol ekranına yazdırıldı **ek1*

7- "deneme.txt" den bilgiler okundu.

8- Okunan bilgilerle huffman ağacı oluştu.

9-Oluşan ağaçlardaki karakterlerin her birinin ikilik gösterimi oluşturuldu.

10-Her bir karakterin ikilik gösterimi ve sıkıştırılan dosya boyutu konsol ekranına yazdırılırdı. **ek2*

11-Program bitti.

V.KULLANILAN KÜTÜPHANELER

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <stdint.h>
```

Ekler

ek1

```
1.Token
offset :0 -> length :0 -> karakter :T

2.Token
offset :0 -> length :0 -> karakter :-

3.Token
offset :0 -> length :0 -> karakter :

4.Token
offset :0 -> length :0 -> karakter :p

5.Token
offset :0 -> length :0 -> karakter :

6.Token
offset :0 -> length :0 -> karakter :d

7.Token
offset :0 -> length :0 -> karakter :
```

```
747.Token
offset :243 -> length :2 -> karakter :s

748.Token
offset :255 -> length :1 -> karakter :z

749.Token
offset :209 -> length :3 -> karakter :-

750.Token
offset :98 -> length :4 -> karakter :-

751.Token
offset :227 -> length :1 -> karakter :y

752.Token
offset :126 -> length :1 -> karakter :r

753.Token
offset :249 -> length :1 -> karakter :z

754.Token
offset :70 -> length :1 -> karakter :


------>Olusturulan token sayisi : 754
----->Metin boyutumuz :2404
------->Sikistirilmis metin boyutumuz :2262
```

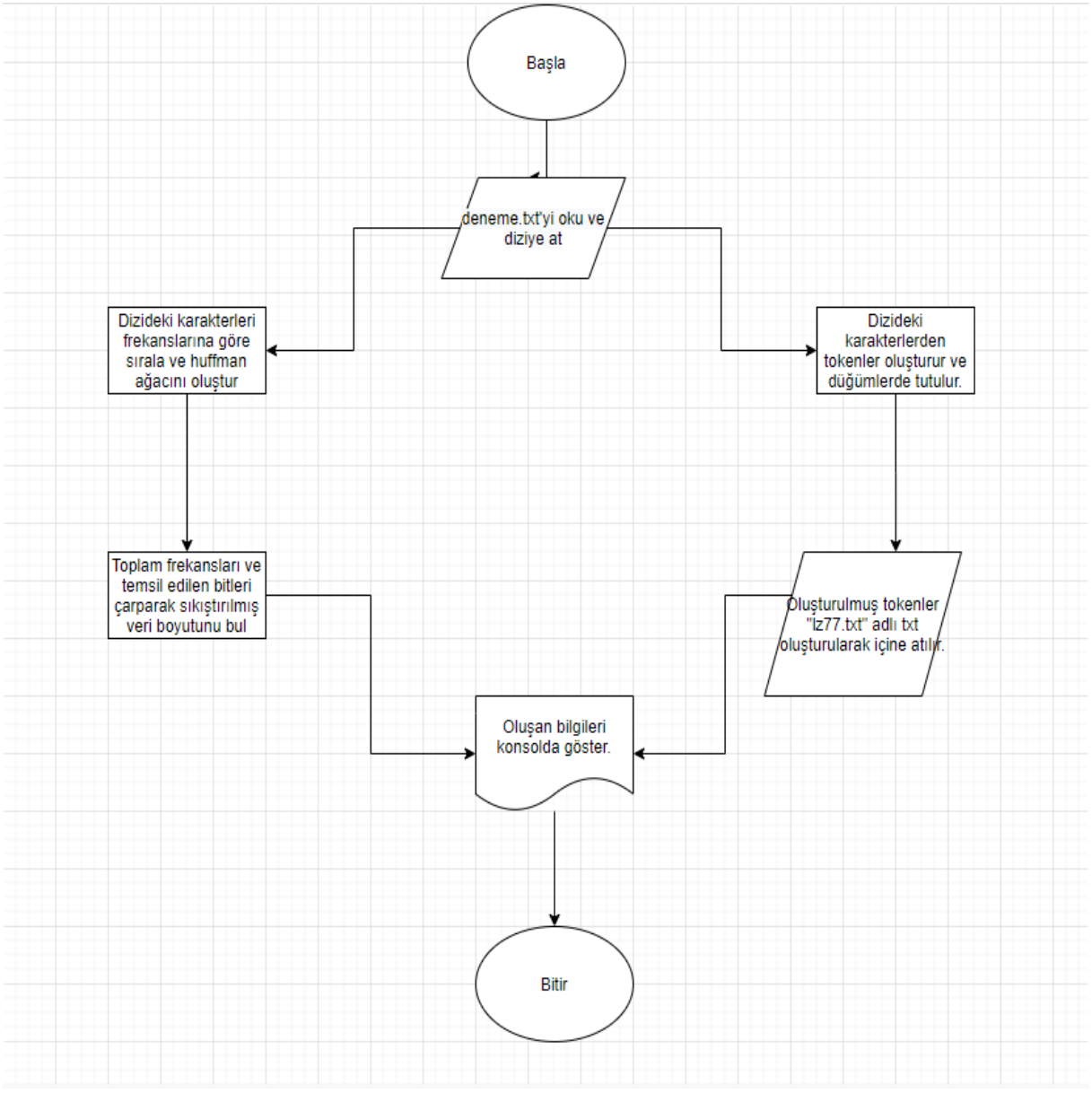
ek2

```
------------HUFFMAN------------

*** m : 00000 : 88
*** ? : 000010000 : 6
*** V : 0000100010 : 3
*** A : 0000100011 : 3
*** p : 00001001 : 11
*** v : 0000101 : 20
*** T : 000011000 : 5
*** f : 000011001 : 5
*** N : 000011010000 : 1
*** D : 000011010001 : 1
*** - : 00001101001 : 1
*** P : 0000110101 : 2
*** B : 0000110110 : 2
*** H : 0000110111 : 2
*** c : 0000111 : 18
*** i : 0001 : 144
***   : 001 : 294
*** y : 01000 : 72
*** k : 01001 : 70
*** l : 0101 : 133
*** b : 01100 : 67
*** d : 01101 : 66
*** S : 0111000000 : 2
*** R : 0111000001 : 2
*** U : 0111000010 : 2
```

```
*** g : 0111001 : 16
*** z : 011101 : 28
*** o : 01111 : 57
*** n : 1000 : 116
*** r : 1001 : 111
*** a : 101 : 222
*** t : 11000 : 53
*** s : 11001 : 51
*** , : 1101000 : 13
*** . : 1101001 : 12
*** h : 110101 : 25
*** u : 11011 : 46
*** e : 111 : 195


***Ana metin boyutu = 19232
***Huffman sonrasi metin boyutu = 8409
```

.

AKIŞ DİYAGRAMI



ALGORITMA PERFORMANS KARŞILAŞTIRMASI

Algoritma Adı	Orijinal Dosya Boyutu	Sıkıştırılmış Dosya Boyutu
LZ77	2404 byte	2262 byte
Huffman	19232 bit	8409 bit

Huffman algoritmasını bir dosyaya yazdırmadığımız için hesaplamayı bit üzerinden yaptık.

Sıkıştırma Oranları

LZ77 - %5,09

Huffman - %56,275

**Oranlara bakıldığı zaman huffman algoritmasını çok daha büyük bir oranda sıkıştırma elde etmiş oldu.*

KAYNAKÇA

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

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

https://zlib.net/feldspar.html

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

https://tools.ietf.org/html/rfc1951

https://www.youtube.com/watch?v=s9whv_QGizM

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

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