YILDIZ TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ
2014-2015 ÖĞRETİM YILI GÜZ YARIYILI

VERİ YAPILARI VE ALGORİTMALAR ÖDEV-5 /LABORATUVAR-2 (BLM-2512/ GRUP:1)

Hazırlanan Anabilim Dalı
Bilgisayar Bilimleri Anabilim Dalı

KONU: HUFMANN İLE SIKIŞTIRILMIŞ VE KODLANMIŞ BİR BİLGİNİN ÇÖZÜLMESİ VE SIKIŞTIRMA VERİMİNİN HESAPLANMASI

Hazırlayan

Mert Sevil
09013057
Bilgisayar Mühendisliği Lisans Programı

Öğretim Üyesi

Prof. M. Yahya KARSLIGİL

İSTANBUL, 2014

1. İçindekiler

2. Özet, ödevin amacı ve kısaca tanıtılması

Ödevde Hufmann ağacı ile sıkıştırılmış bir metinin anlamsız sayı karşılıklarından anlamlı ve şifresi çözülmüş metinler elde edilmesine yönelik algoritma incelenmiştir. Bu amaçla öncelikle el ile Hufmann ağacı çizilmiş, metin el ile çözülmek istenmiştir. Verim el ile matematiksel formüller ile bulunmuştur. Ardından algoritması oluşturulan yazılım DEV C++ derleyicisinde C dilinde kodlanmıştır. Analizleri el ile gerçekleştirilen sonuçlarla karşılaştırılan algoritma sonuçları incelenmiş ve analiz edilmiştir.

Ödevin amacı algortima tasarlamayı kavramaktır. Sıkıştırma algoritmasının incelenerek veriminin gözlenmesi, daha az bellek alanında daha çok veri tutabilme hatta kriptoloji alanında hala geçerliği olan Hufmann'ın gerçekleştirilerek teorik bilginin pratik olarak uygulanması imkânı sunar. Bu açıdan verimli ve optimize edilmiş bir algoritma tasarımının yanı sıra bu algoritmaların bir programlama dilinde kodlanabilme beceresini de geliştirmiştir. Bu açıdan ödev mühendislik eğitimi açısından önemli bir köşetaşı olarak görülmelidir.

Hufmann algoritması ile teorik bilgi verilmek istenirse;

Huffman Kodu, Bilgisayar biliminde, veri sıkıştırması için kullanılan, bir entropi kodlama algoritmasıdır. David A. Huffman tarafından 1952 yılında geliştirilmiştir. [1]

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. [1]

Huffman tekniği günümüzde tek başına kullanılmaz. LZW, RLE gibi yöntemlerle birlikte kullanılır. [1]

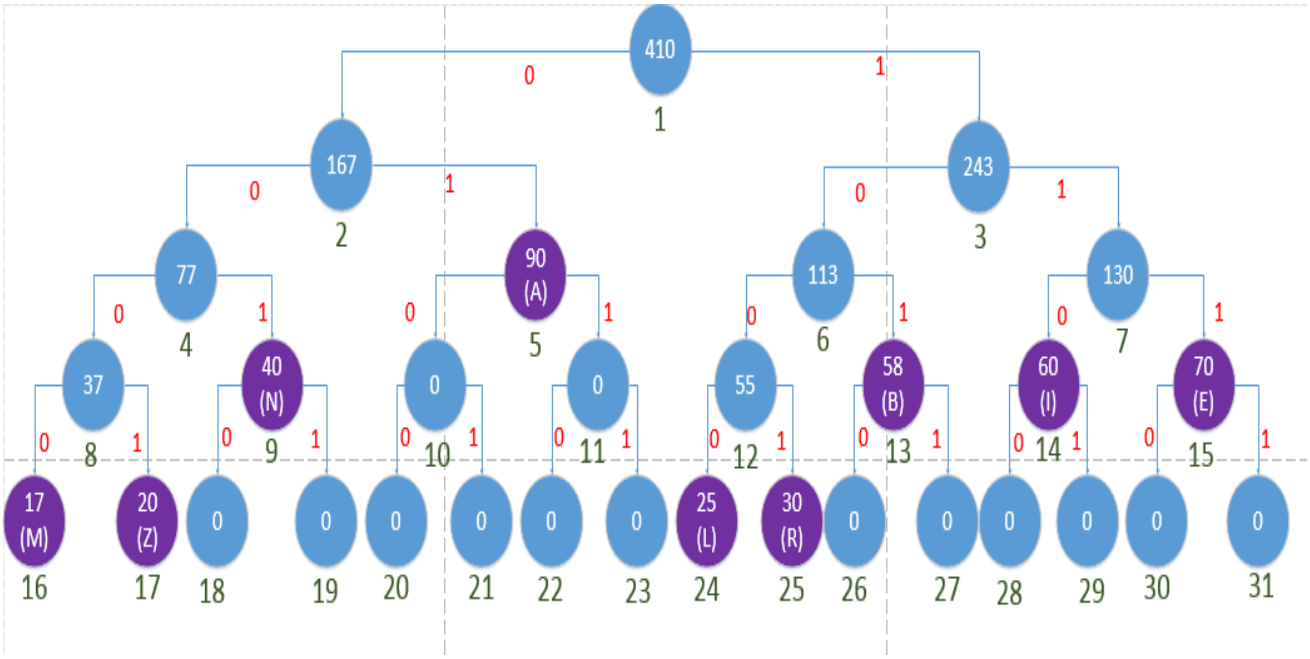
Huffman'ın algoritması, veri içerisindeki karakterlerin kullanım sıklığına (frekans) göre bir ağaç oluşturur. Ağacın en tepesinden aşağıya doğru ilerlerken sola ayrılan dal için 0, sağa ayrılan dal için 1 kodu verilir. [1]

Dezavantajları: Huffman algoritması az sayıda karakter çeşidine sahip ve büyük boyutlardaki verilerde çok kullanışlı olabilir. Fakat oluşturulan ağacın sıkıştırılmış veriye eklenmesi zorunludur. Bu da sıkıştırma verimini düşürür. Adaptive Huffman gibi teknikler bu sorunu halletmek için geliştirilmişlerdir. [1]

3. Hufmann ağacının oluşturulması

Visio programı ile Hufmann ağacı çizilmiştir. Buna göre;

Mor renk: Yaprak olup karekteri tanıyan noktadır.
Kırmızı renk: 0'lar küçük olan sol adresleri, 1'ler büyük olan sağ adresleri gösterir
Yeşil renk: Düğüm adresini gösterir
Mavi renk: Yaprak olmayan ağaç gözlerini tanımlar
Harfler: Yapraklardaki sayıların karekter karşılıklarını tanımlar
Beyaz renkli sayılar: Frekans değerini belirtir



Şekil-1 Hufmann ağacının çizilmesi

4. Verilerin el ile oluşturulması

Visio ile çizilen Hufmann ağacından da görüleceği gibi karekterlere ait kod karşılıkları rahatlıkla belirlenebilir. Bu durum Tablo 1'de belirlenmiştir.

Karekter	Kod Karşılığı	Frekans	Hufmann Düğüm Adresi
A	01	90	5
N	001	40	9
B	101	58	13
I	110	60	14
E	111	70	15
M	0000	17	16
Z	0001	20	17
L	1000	25	24
R	1001	30	25

Tablo-1 Karakterler ve kod karşılıkları

Buna göre kod el ile çözülebilir. Bu sonuç daha sonra bilgisayar ile elde edilen sonuçlar ile karşılaştırılacaktır.

01	**A**
001	**N**
001	**N**
111	**E**
1000	**L**
111	**E**
1001	**R**
110	**I**
0000	**M**
110	**I**
0001	**Z**

Kodun ayrıştırılmamış hali:

0 1 0 0 1 0 0 1 1 1 1 1 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0

5. Verimin el ile bulunması

Verimin bulunması için;

Sıkıştırılmadan gönderilen veri için bit sayısı:

$$\sum_{i=1}^{Karekter\ sayisi} frekansi * 8$$

$$= 8 * 90 + 40 * 8 + 40 * 8 + 8 * 70 + 25 * 8 + 70 * 8 + 30 * 8 + 60 * 8 + 17 * 8 + 60 * 8 + 20 * 8$$

Sıkıştırılmadan önceki bit sayısı= 4176

Sıkıştırıldıktan sonraki haliyle gönderilen veri için bit sayısı:

$$\sum_{i=1}^{Karekter\ sayisi} frekansi * karektereaitbitsayisi$$

$$= 2 * 90 + 40 * 3 + 40 * 3 + 3 * 70 + 25 * 4 + 70 * 3 + 30 * 4 + 60 * 3 + 17 * 4 + 60 * 3 + 20 * 4$$

Sıkıştırılmadan önceki bit sayısı= 1568

Verim için aşağıdaki yapı kullanılabilir;

$$Verim = \frac{Sıkıştırılmadanönce - Sıkıştırılınca}{Sıkıştırılmadanonce}$$

$$Verim = \frac{4176 - 1568}{4176} = 0.6245 = \%62.45$$

6. Algoritmanın aktarılması

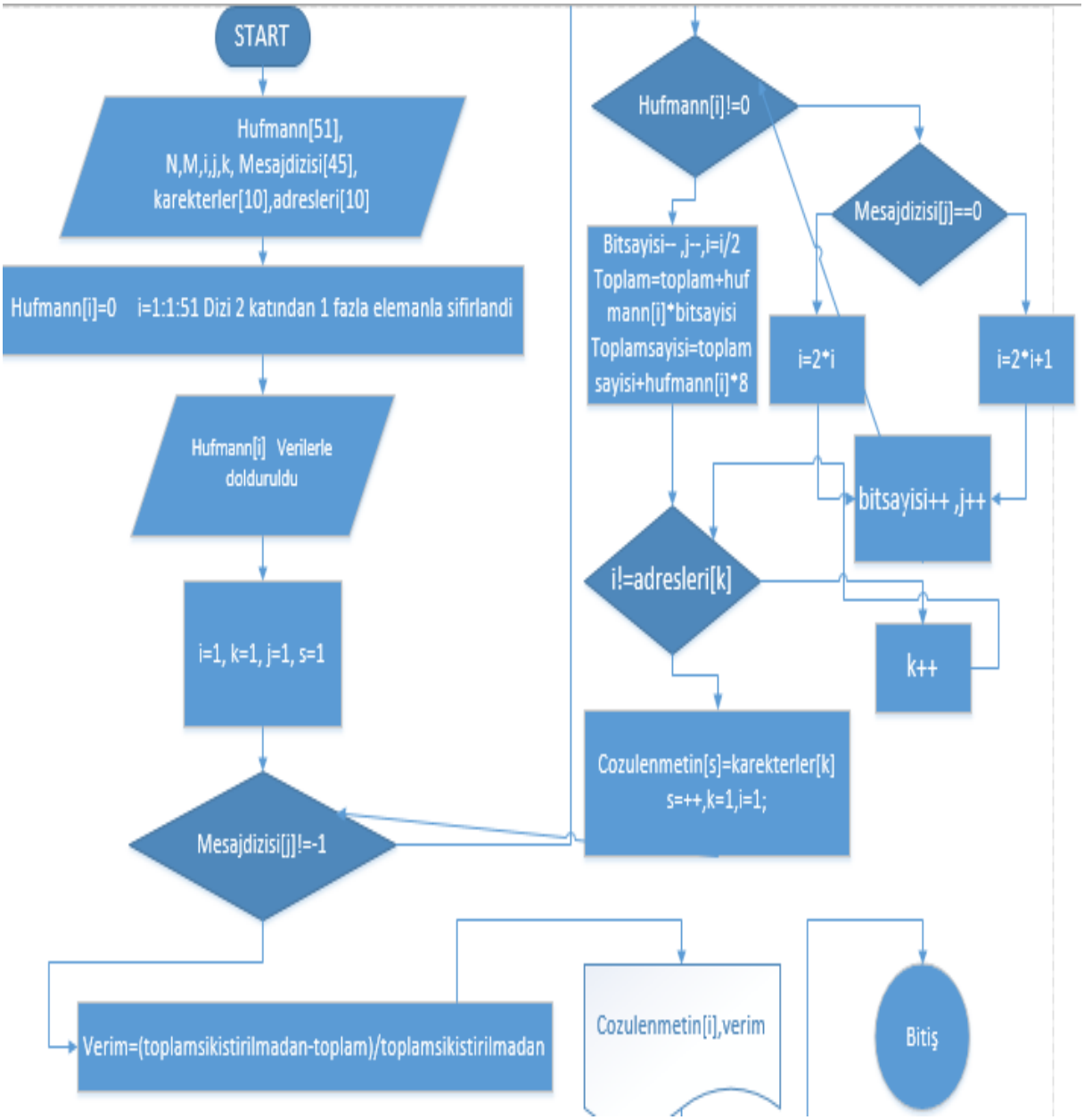
Visio ile çizilen algoritma şekilde görülmektedir.

Buna göre içiçe iki döngü algoritmanın temel çalışma noktasını oluşturur. Birinci döngü karakterlerin bittiğinin anlaşılmasına ait döngüdür. Bu birkaç yolla yapılabilir. Algortimamda daha önce mesaj dizisinin son elemanından sonraki elemanı -1 ile doldurdum. Bu mesaj karakterlerinin bittiğini göstermektedir. Bu mantıkla durma koşulu mesajdizisinin j. gözü -1'i görünceye kadar algoritma işletilmelidir.

İçiçe döngünün ikinci döngüsü hufmann ağacında ilerlemektir. Burada ağaçta 0 gözü görülünceye kadar ilerlenir. Bu nedenle Hufmann[i]!=0 denilerek While içinde dönülür. İ gözü mesajdizisinin[j] gözüne bakılarak 2 katına yâda 2 katından bir fazlasına işaret edilir.

Karakter bulununca While döngüsünden çıkılmış olur. Bu açıdan i adresi yarıya bölünmelidir. Bulunan i değeri adresler dizisinde taranarak adresi bulunur. Bulunan k adresi harfeler dizisindeki adresi işaret ederek harf karşılığı bulunur. Bu çözülen metin dizisine atanır. Değerler bir yapılarak ilk döngü içinde yeniden dönülür, bütün karakterler için aynı algoritma gerçekleştirilir. Son olarak başta da belirtildiği gibi -1 değeri görülünce döngüden çıkılır.

Sonuçta döngüden çıkılınca karekterler bulunmuş olur ve yazdırılır.



7. Yazılan C kodunun belirtilmesi

C programla dili ile gerçekleştirilen algoritma için Dev C++ derleyicisi kullanılmıştır.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>


int main(){
```

```
        int
hufmann[51],N,M,i,Mesajdizisi[100],adresleri[10],k=0,s=0,j=0,Toplam=0,bitsayisi=0,Toplamsi
kistirilmadan=0; // toplam= verimi bulmak icin sikistirma islemi ile olusan bit sayisini tutar
        //toplamsikistirilmadan toplam bit sayisini tutar sikistirilmadan, bitsayisi her karekter
icin hufmann ile kac bit yer tutuldugunu sayar
        //adresleri dizisi adresi verilen karekterlerin hufmann agacindaki adreslerini tutan
dizidir
        //s cozulen metin icin tanımlanan indis dizi degiskeni
        //k hufmannda bulunan i indisinin adres degerinin hangi karektere denk geldigini
bulmak icin tarama  yaparken kullanılan dizi degiskeni
        //j= mesaj dizisinden 0-1 leri cekmek icin kullanılır
        //i hufmann agacinda dolasmak icin kullanılır
        //mesaj dizisi verilen 0-1 lerden olusan dizidir
        //N Hufmanin eleman sayisi
        //M iki kati kadar 0lamak için kullanılan eleman sayisidir

        char karekterler[10],cozulenmetin[50]; //karekterler verilen karekterleri tutar,
cozulenmetin ise sonucta bulunmasi istenilen decode edilen metindir
        float verim=0; // verimin hesaplanmasi

        M=25 ;// Huffman treenin eleman sayisi
        N=2*M+5 ; // Hufmanin gözlerini sifirlamak icin

        for(i=1;i<N;i++){       //Hufmann dizisi 2 kati elemandan fazlasiyla sifirlandi
                hufmann[i]=0;
        }

        hufmann[1]=410;         //HUFMANN ELLE OLUSTURULDU
        hufmann[2]=167;
        hufmann[3]=243;
        hufmann[4]=77;
        hufmann[5]=90;
        hufmann[6]=113;         //HUFMANN ELLE OLUSTURULDU
        hufmann[7]=130;
        hufmann[8]=37;
        hufmann[9]=40;
        hufmann[10]=0;
        hufmann[11]=0;
        hufmann[12]=55;         //HUFMANN ELLE OLUSTURULDU
        hufmann[13]=58;
        hufmann[14]=60;
        hufmann[15]=70;
        hufmann[16]=17;
        hufmann[17]=20;
        hufmann[18]=0;          //HUFMANN ELLE OLUSTURULDU
        hufmann[19]=0;
        hufmann[20]=0;
```

```
hufmann[21]=0;            //HUFMANN ELLE OLUSTURULDU
hufmann[22]=0;
hufmann[23]=0;
hufmann[24]=25;
hufmann[25]=30;           //HUFMANN ELLE OLUSTURULDU

printf("Verilenler:\n");
printf("\n1-Hufmann agaci\n");
          for(i=1;i<N;i++){         //Hufmann dizisi yazdiriliyor
          printf("%d  ",hufmann[i]);
}


Mesajdizisi[1]=0;     // 0 ve 1 ile kodlanmis dizinin olusturulmasi //A KAREKTERİ
Mesajdizisi[2]=1;

Mesajdizisi[3]=0;    //N KAREKTERİ
Mesajdizisi[4]=0;
Mesajdizisi[5]=1;

Mesajdizisi[6]=0;  //N KAREKTERİ
Mesajdizisi[7]=0;
Mesajdizisi[8]=1;

Mesajdizisi[9]=1;     // 0 ve 1 ile kodlanmis dizinin olusturulmasi    //E KAREKTERİ
Mesajdizisi[10]=1;
Mesajdizisi[11]=1;

Mesajdizisi[12]=1;    //L KAREKTERİ
Mesajdizisi[13]=0;
Mesajdizisi[14]=0;
Mesajdizisi[15]=0;
                       // 0 ve 1 ile kodlanmis dizinin olusturulmasi
Mesajdizisi[16]=1;    //E KAREKTERİ
Mesajdizisi[17]=1;
Mesajdizisi[18]=1;

Mesajdizisi[19]=1;   //R KAREKTERİ
Mesajdizisi[20]=0;
Mesajdizisi[21]=0;
Mesajdizisi[22]=1;
                        // 0 ve 1 ile kodlanmis dizinin olusturulmasi
Mesajdizisi[23]=1;      //I KAREKTERİ
Mesajdizisi[24]=1;
Mesajdizisi[25]=0;

Mesajdizisi[26]=0;     //M KAREKTERİ
```

```
Mesajdizisi[27]=0;
Mesajdizisi[28]=0;   // 0 ve 1 ile kodlanmis dizinin olusturulmasi
Mesajdizisi[29]=0;

Mesajdizisi[30]=1;    //I KAREKTERİ
Mesajdizisi[31]=1;
Mesajdizisi[32]=0;

Mesajdizisi[33]=0;     // 0 ve 1 ile kodlanmis dizinin olusturulmasi //Z KAREKTERİ
Mesajdizisi[34]=0;
Mesajdizisi[35]=0;
Mesajdizisi[36]=1;



Mesajdizisi[37]=-1;
Mesajdizisi[38]=-1;
Mesajdizisi[39]=-1;
Mesajdizisi[40]=-1;   //Bitiş noktasini gosteriyor diger 0'larla karismasin diye
Mesajdizisi[41]=-1;
Mesajdizisi[42]=-1;
Mesajdizisi[43]=-1;
Mesajdizisi[44]=-1;
printf("\n");
printf("\n2-0 ve 1lerden olusan ve cozulmesi beklenen mesaj bilgisi\n");

                for(i=1;i<36;i++){         //Mesaj dizisi dizisi yazdiriliyor
        printf("%d  ",Mesajdizisi[i]);
}

karekterler[1]='A';     adresleri[1]=5;
karekterler[2]='N';     adresleri[2]=9;
karekterler[3]='B';     adresleri[3]=13;
karekterler[4]='I';     adresleri[4]=14;
karekterler[5]='E';     adresleri[5]=15;
karekterler[6]='M';     adresleri[6]=16;
karekterler[7]='Z';     adresleri[7]=17;
karekterler[8]='L';     adresleri[8]=24;
karekterler[9]='R';     adresleri[9]=25;

printf("\n");
printf("\n3-Karekterler ve hufmann agacindaki adresleri\n");

                for(i=1;i<10;i++){         //Mesaj dizisi dizisi yazdiriliyor
        printf("%d  ",adresleri[i]);
}
                printf("\n");
```

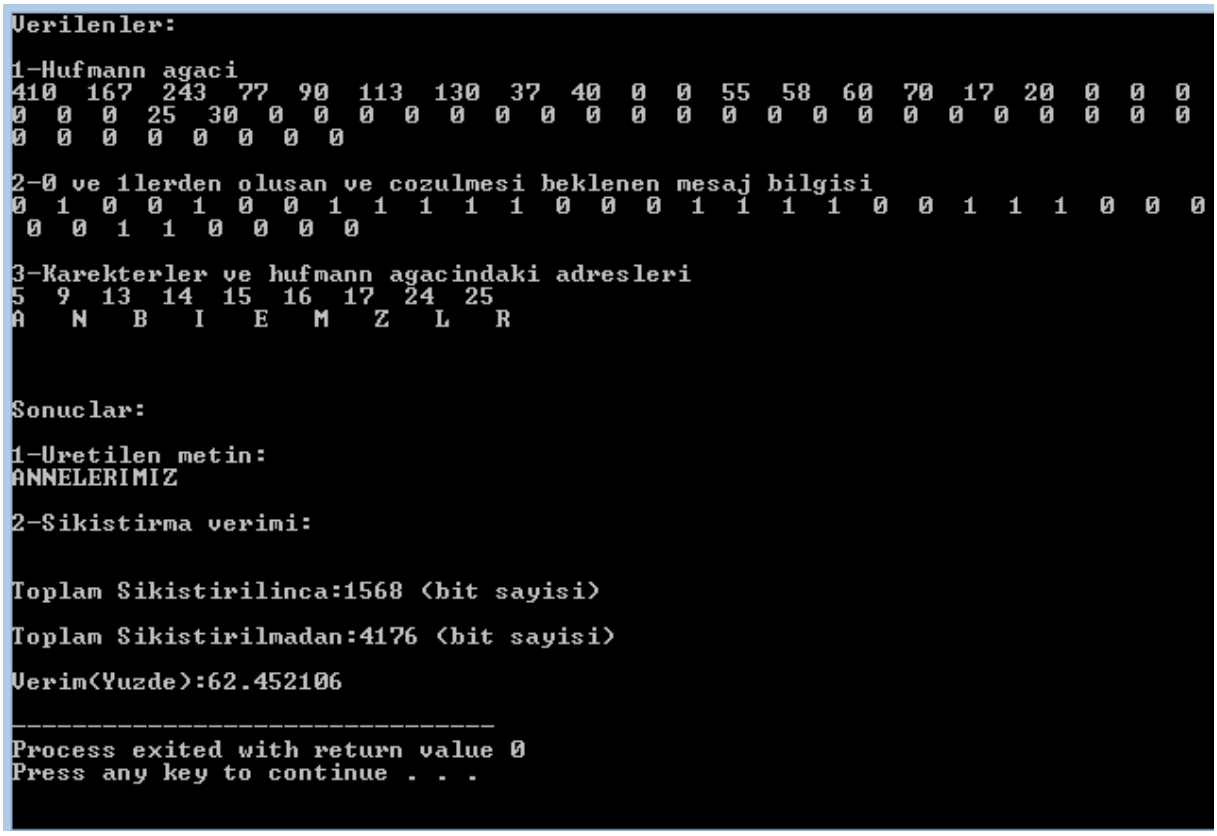
```
                    for(i=1;i<10;i++){          //Mesaj dizisi dizisi yazdiriliyor
                printf("%c   ",karekterler[i]);
        }

        //Cozum icin gelistirlen algoritma baslangici
        i=1;
        k=1;
        s=1;
        j=1;
        printf("\n");
        while(Mesajdizisi[j]!=-1){
        //printf("i:%d\n",i); //Analiz için yazdirma
        while(hufmann[i]!=0){
                //printf("Dongu İci:Mesajdizisi[i]:%d\n",Mesajdizisi[j]);  //Analiz için yazdirma
                if(Mesajdizisi[j]==0)
                i=2*i;
                else
                i=2*i+1;

                bitsayisi++;
                j++;
        //      printf("Dongu İci:j:%d\n",j);  //Analiz için yazdirma
        }
        bitsayisi--;
                //printf("\ni:%d\n",bitsayisi); //Analiz icin yazdirma
        j--;
        i=i/2;
        Toplam=Toplam+hufmann[i]*bitsayisi;
        Toplamsikistirilmadan=Toplamsikistirilmadan+hufmann[i]*8;
        bitsayisi=0;
        //      printf("\nToplam:%d\n",Toplam);    // Analiz için yazdirma
        //              printf("\nToplamsikistirilmadan:%d\n",Toplamsikistirilmadan);
//Analiz için yazdirma
//      printf("\ni:%d\n",i);  //Analiz için yazdirma
//      printf("j:%d\n",j);   //Analiz için yazdirma
        while(i!=adresleri[k])
        k++;


//      printf("k:%d\n",k);    //Analiz için yazdirma
        cozulenmetin[s]=karekterler[k];

        s++;
        // printf("s:%d\n",s); //Analiz için yazdirma
//       printf("%c\n",cozulenmetin[s-1]);  //Analiz için yazdirma
        k=1;
        i=1;
```

```
    }
     // printf("%d",s); //Analiz için yazdirma
    printf("\n");
    printf("\n");
    printf("\nSonuclar:\n");
    printf("\n");
          printf("1-Uretilen metin:\n");
          for(i=1;i<s;i++){          //Sonuc dizisi
          printf("%c",cozulenmetin[i]);
          }

          printf("\n");
    printf("\n");
          printf("2-Sikistirma verimi:\n");
          printf("\n");
             printf("\nToplam Sikistirilinca:%d (bit sayisi)\n",Toplam);     // Analiz için yazdirma
                   printf("\nToplam Sikistirilmadan:%d (bit sayisi)\n",Toplamsikistirilmadan);
//Analiz için yazdirma
             verim=(float(float(Toplamsikistirilmadan-Toplam))/(float
(Toplamsikistirilmadan)))*100 ;
             printf("\nVerim(Yuzde):%f\n",verim);


          return 0;
          getch();
}
```

8. Sonuçlar ve analizi

Sonuçlar kısmında kodun çalıştırılmasına dair ekran çıktısı incelenmiştir. Kod çalıştırılınca karşımıza çıkan ekranda öncelikle verilenler gözlenmektedir. Hufmann ağacı, mesaj dizisi ve karakterler ve o karakterlerin hufmann ağacındaki adreslerine karşılık düşen adres sayıları verildiği gibi açıkça ve kullanıcının anlayacağı şekilde yazdırılmıştır. Sonuçlar kısmında ise mesaj dizisi olarak kodlanan 0-1'li dizinin çözülen metinsel karşılığı bulunuştur. Ayrıca istenilen verimde hesaplanarak bulunarak eklenmiştir.

Algoritma analizi sonuçların doğruluğu üzerinden incelenmiştir. El ile hesaplanan değeler 4. ve 5. bölümde bulunmuştur. 4. bölümde bulunan ANNELERIMIZ ile algoritma sonrası bulunan değer aynı çıkmıştır. 5. bölümde bulunan verim değeri ise 62.45 olarak bulunmuştur. Bu değere algoritmanın koşturulması sonucu üretilen değerle birebir olarak örtüşmektedir.



Böylece ödev analizi de yapılmak suretiyle sonlandırılmıştır.

9. Kaynakça

[1] http://tr.wikipedia.org/wiki/Huffman_kodu (Erişim Tarihi: 14.11.2014)

Tarih:15.11.2014