HUFFMAN

Huffman algoritması, bir veri kümesinde daha çok rastlanan sembolü daha düşük uzunluktaki kodla, daha az rastlanan sembolleri daha yüksek uzunluktaki kodlarla temsil etme mantığı üzerine kurulmuştur. Bir örnekten yola çıkacak olursak : Bilgisayar sistemlerinde her bir karakter 1 byte yani 8 bit uzunluğunda yer kaplar. Yani 10 karakterden oluşan bir dosya 10 byte büyüklüğündedir. Çünkü her bir karakter 1 byte büyüklüğündedir. Örneğimizdeki 17 karakterlik veri kümesi "ALGORITMAANALIZI" olsun. "A" karakteri çok fazla sayıda olmasına rağmen "G,O,R,T,M,N,Z" karakterleri tektir. Eğer bütün karakterleri 8 bit değilde veri kümesindeki sıklıklarına göre kodlarsak veriyi sembolize etmek için gereken bitlerin sayısı daha az olacaktır.

Bu arada huffman algoritması Binary arama ve heap algoritmalarını barındırır.

A0,I1,L10,G11,O100,R101,T110,M111,N1000,Z1001 kodunu kullanabiliriz. Bu durumda 10 karakterlik verimizi temsil etmek için

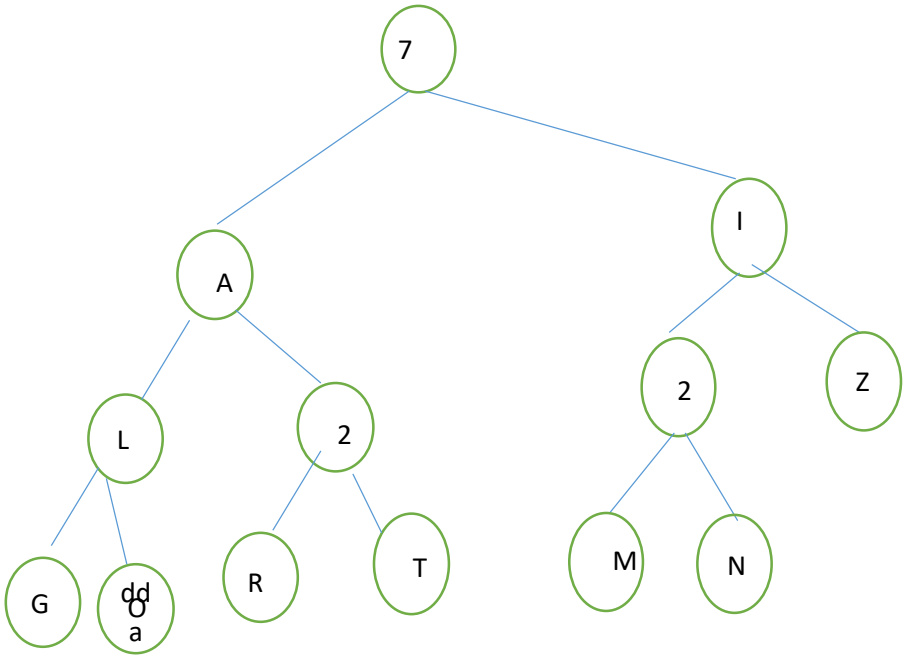
(A kodundaki bit sayısı)*(verideki A sayısı) + (I kodundaki bit sayısı)*(verideki I sayısı) + (L kodundaki bit sayısı)*(verideki L sayısı) + (G kodundaki bit sayısı)*(verideki G sayısı) + (O kodundaki bit sayısı)*(verideki O sayısı) + (R kodundaki bit sayısı)*(verideki R sayısı) + (T kodundaki bit sayısı)*(verideki T sayısı) + (M kodundaki bit sayısı)*(verideki M sayısı) + (N kodundaki bit sayısı)*(verideki N sayısı) + (Z kodundaki bit sayısı)*(verideki Z sayısı) = 1*4 + 1*3 + 2*2+ 2*1+ 3*1+ 3*1+ 3*1 + 3*1+ 4*1+ 4*1= 33 bit

gerekecektir. Halbuki bütün karakterleri 8 bit ile temsil etseydik 16*10 = 160 bite ihtiyacımız olacaktı.

ALGORİTMAANALİZİ KARAKTER DİZİSİNİN FREKANS TABLOSU:

KARAKTERLER	KARAKTER TEKRAR SAYISI (FREKANS)
A	4
I	3
L	2
G	1
O	1
R	1
T	1
M	1
N	1
Z	1

ALGORİTMAANALİZİ KARAKTER DİZİSİNİN HUFFMAN AĞACI:



KARMAŞIKLIK ANALİZİ:

Aşağıdaki kodların fotoğraflarında da detaylı bir şekilde görüldüğü gibi çalışma zamanı analizini yaptığım Huffman algoritmasında karmaşıklığı **O(karakterSayısı)** çıktı.Yani bir karakter dizisinde veya bir cümledeki karakter sayısının değişkenliğine göre algoritmamızın çalışma hızında farklılık gözlemleyeceğiz.

Şimdi ise aşağıdaki fotoğraflar üzerinden daha detaylı bir çalışma zamanı analizi yapalım.Bu analizi yaparken graf renklendirme algoritmasından farklı olarak karakter sayımız üzerinde çalışacağız.

```
1  import java.util.*;
2
3  abstract class HuffmanAgacı implements Comparable<HuffmanAgacı> {
4      public final int sıklık; //------------------------------------------>1
5
6      public HuffmanAgacı(int frekans) {
7          sıklık = frekans; //---------------------------------------------->1
8      }
9
10
11     @Override
12     public int compareTo(HuffmanAgacı tree) {
13         return sıklık - tree.sıklık;//----------------------------------->1
14     }
15 }//------------------------------------------------------------------------>TOPLAM = 3
16
17 class HuffmanYaprak extends HuffmanAgacı {
18     public final char deger;//--------------------------------------------->1
19
20     public HuffmanYaprak(int freq, char val) {
21         super(freq);//----------------------------------------------------->karakterSayısı
22         deger = val;//----------------------------------------------------->karakterSayısı
23     }
24 }
25  //------------------------------------------------------------------------>TOPLAM =2*karakterSayısı+1
26 class HuffmanDügüm extends HuffmanAgacı {
27     public final HuffmanAgacı sol, sag; //--------------------------------->1
28
29     public HuffmanDügüm(HuffmanAgacı l, HuffmanAgacı r) {
30         super(l.sıklık + r.sıklık);//------------------------------------->2*karakterSayısı
31         sol = l;//-------------------------------------------------------->2*karakterSayısı
32         sag = r;//-------------------------------------------------------->2*karakterSayısı
```

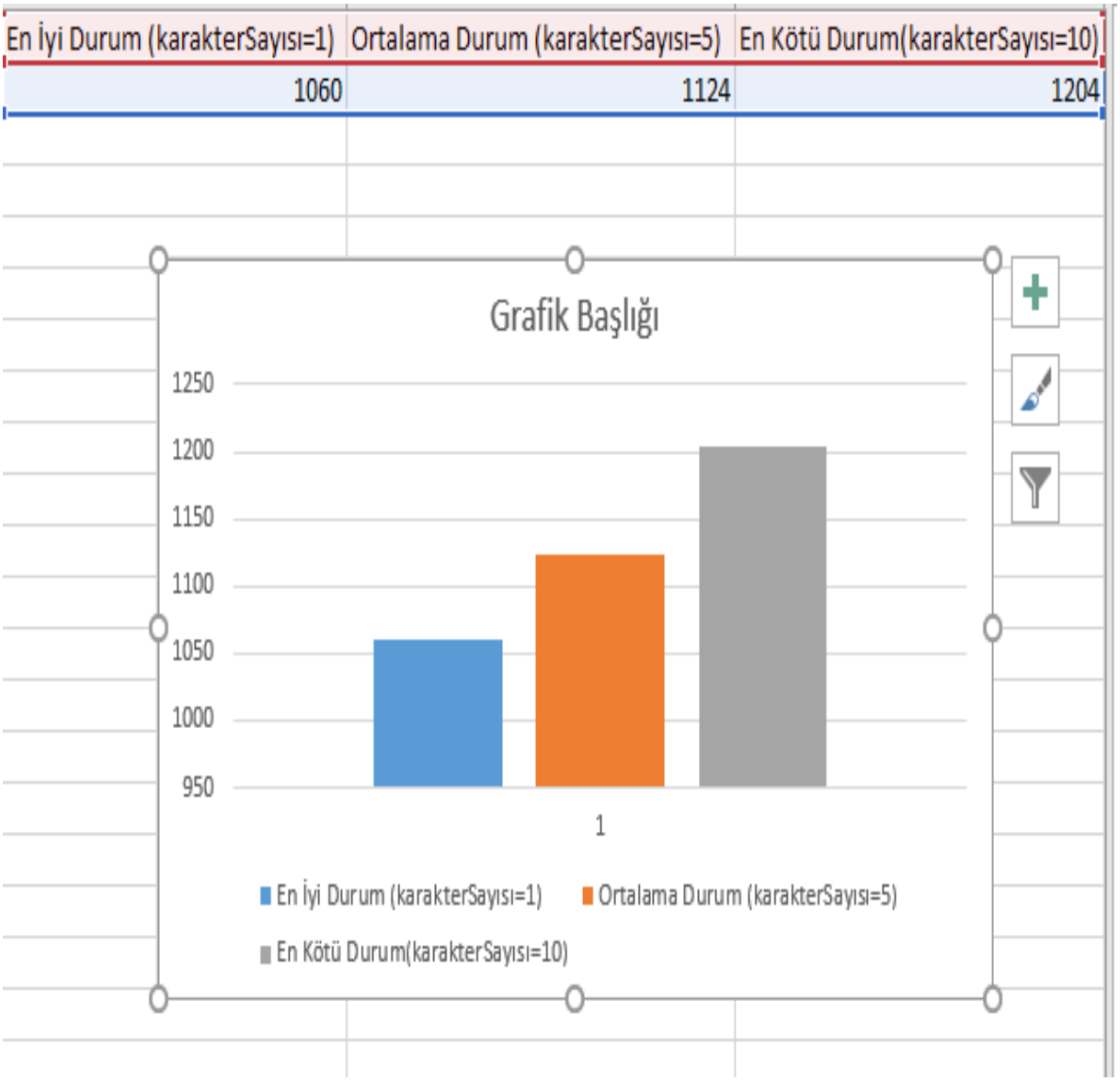
```
32              sag = r;//------------------------------------------------------>2*karakterSayısı
33          }
34      }//-------------------------------------------------------------------------->TOPLAM = 6*karakterSayısı+1
35      public class HuffmanKodu {
36
37          public static HuffmanAgacı huffmanAgacı(int[] karakterFrekansları) {
38          PriorityQueue<HuffmanAgacı> agac = new PriorityQueue<HuffmanAgacı>();//-->2*karakterSayısı
39
40
41              for (int i = 0; i < karakterFrekansları.length; i++){//--------------->255
42                  if (karakterFrekansları[i] > 0){//---------------------------->karakterSayısı
43              agac.offer(new HuffmanYaprak(karakterFrekansları[i], (char)i));//--->karakterSayısı
44                  }
45              assert agac.size() > 0;//------------------------------------------->255
46              }
47              while (agac.size() > 1) {//----------------------------------->karakterSayısı
48
49                  HuffmanAgacı a = agac.poll();//------------------------------>karakterSayısı
50                  HuffmanAgacı b = agac.poll();//------------------------------>karakterSayısı
51
52
53                  agac.offer(new HuffmanDügüm(a, b));//------------------------>karakterSayısı
54              }
55              return agac.poll();//---------------------------------------------->1
56          }//-------------------------------------------------------------------------->TOPLAM = 8*karakterSayısı+511
57
58          public static void koduYazdırma(HuffmanAgacı agac, StringBuffer prefix) {
59              assert agac != null;//--------------------------------------------->1
60              if (agac instanceof HuffmanYaprak) {//----------------------------->1
61                  HuffmanYaprak yaprak = (HuffmanYaprak)agac;//---------------->1
62
63                  System.out.println(yaprak.deger + "\t" + yaprak.sıklık + "\t" + prefix);//-------->1
64
65              } else if (agac instanceof HuffmanDügüm) {//------------------------>1
66                  HuffmanDügüm node = (HuffmanDügüm)agac;//---------------------->1
67
68
69                  prefix.append('0');//---------------------------------------->1
70                  koduYazdırma(node.sol, prefix);//---------------------------->1
71                  prefix.deleteCharAt(prefix.length()-1);//---------------------->1
72
73
74                  prefix.append('1');//---------------------------------------->1
75                  koduYazdırma(node.sag, prefix);//---------------------------->1
76                  prefix.deleteCharAt(prefix.length()-1);//---------------------->1
77              }
78          }//-------------------------------------------------------------------------->TOPLAM = 12
79
80          public static void main(String[] args) {
81              String test = "ALGORITMAANALIZI";//---------------------->1
82
83              int[] charFrekans = new int[255];//---------------------->1
84
85              for (char c : test.toCharArray()){//---------------------->255
86
87                  charFrekans[c]++;//---------------------------------->255
88              }
89              HuffmanAgacı agac = huffmanAgacı(charFrekans);//------------>1
90
91              System.out.println("Karakter|\tTekrar|\tHUFFMAN Kodu");//-->1
92              koduYazdırma(agac, new StringBuffer());//-------------------->1
93              System.out.println("Çalış zamanı = "+System.nanoTime());//--->516
94              //------------------------------------------------------------>TOPLAM = 16*karakterSayısı+1044
```

Kısaca açıklamak gerekirse huffmanAgacı metodu genel kontrol metodumuz bu metod üzerinden diğer bütün metodlara ve değişkenlere erişe biliyoruz. Demek oluyor ki bu metod daki bütün döngüleri iyi takip etmek gerekirir çünkü bu metod üzerinden ve döngülerin içinde HuffmanAgacı sınıfımızdan bir çok kez nesne oluşturuyor ve for döngümüzün içinde huffmanYaprak metodumuza karaktere sayımız kadar erişiyoruz.

Main metodumuzda algoritmayı çalıştırdıktan sonra yazdığım koddada detaylıca belirttiğim gibi genel karmaşıklığı yani f(karakterSayısı) ,**T(karakterSayısı) = 16*karakterSayısı+1044** olarak alıyor.

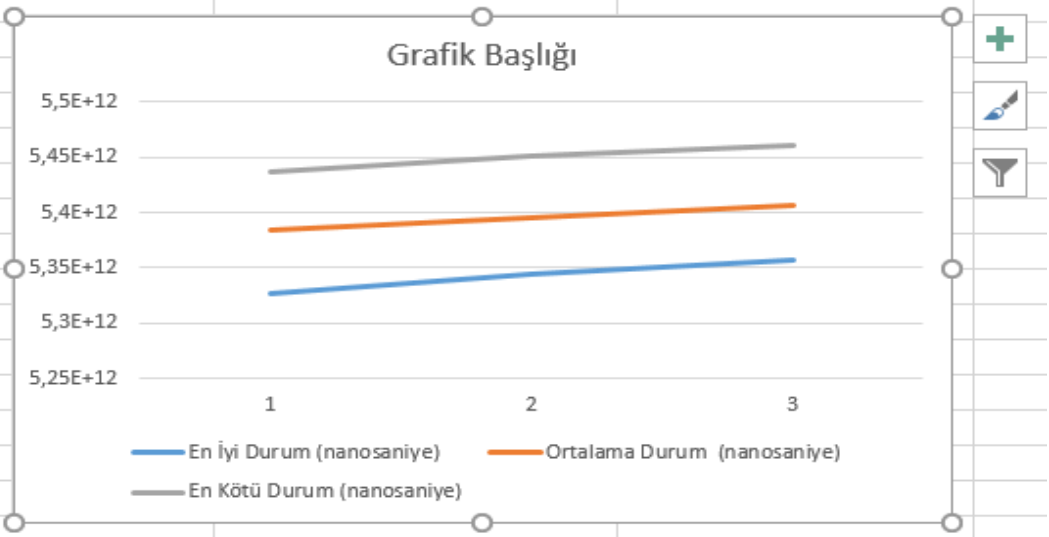
Buradaki karakter sayısı "ALGORITMAANALIZI" karakter dizine bağlı olarak 10 (tekrak eden karakterler 1 kere sayılır.).Bu değer azaldıkça ve arttıkça en iyi durum,en kötü durum ve ortalama durum çalışma zamanı değerlerini alabiliriz şimdi ise aşağıda çalışma zamanı grafikleri üzerinden analiz edelim.

T(karakterSayısı) = 16*karakterSayısı+1044 Denklemi ile Çalışma Zamanı Grafiği:

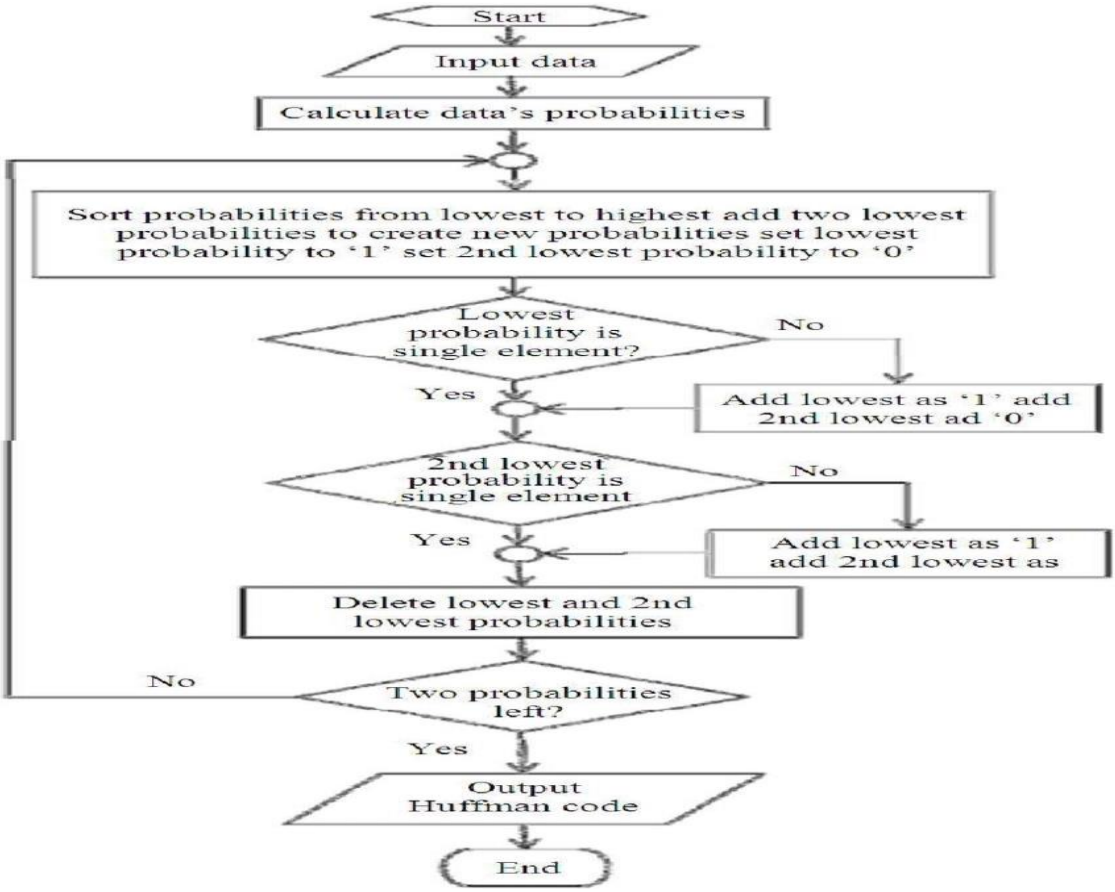


Java da üçer kere Kod (System.nanoTime()) ile Çalışma Zamanı Grafiği:

En İyi Durum (nanosaniye)	Ortalama Durum (nanosaniye)	En Kötü Durum (nanosaniye)
5,3267E+12	5,38399E+12	5,43722E+12
5,34435E+12	5,39472E+12	5,45065E+12
5,35761E+12	5,40618E+12	5,46069E+12



AKIŞ DİYAGRAMI



HAZIRLAYAN : ÖMER CENGİZ 16260056