

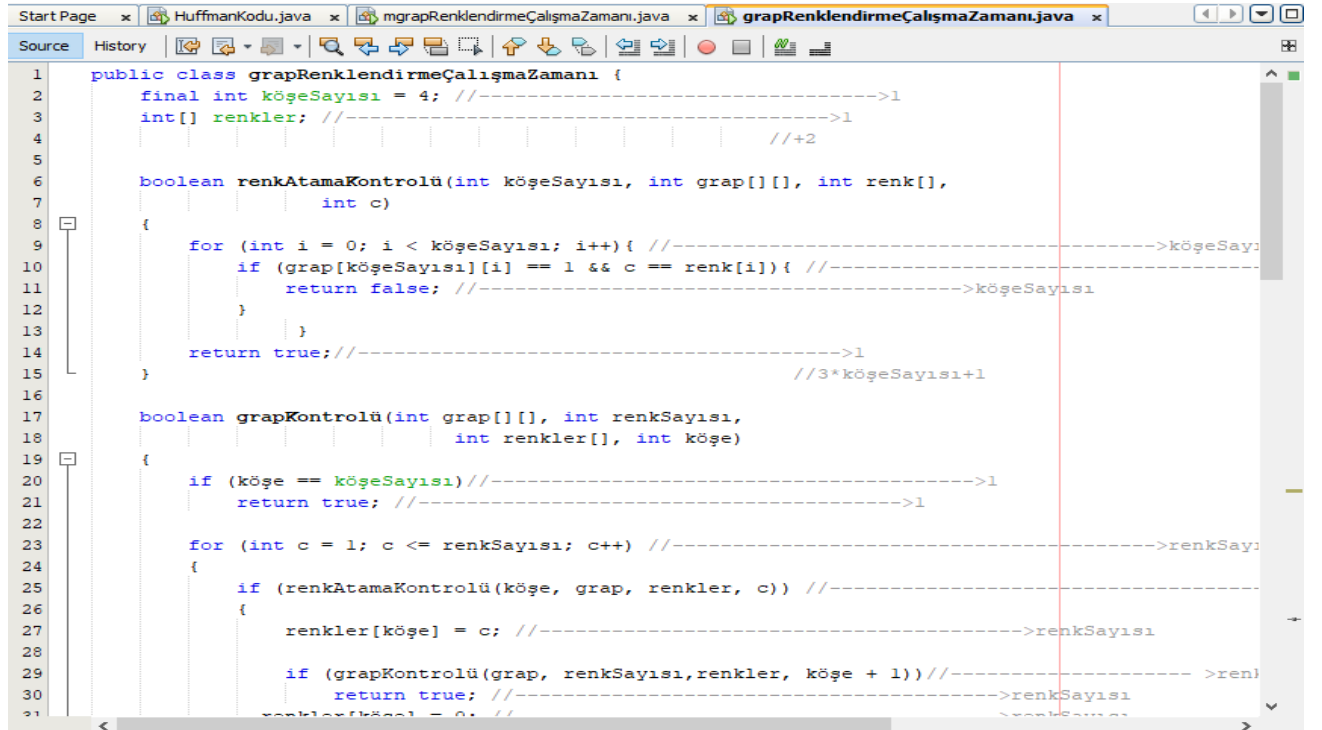
## Graf Renklendirme Problemi

Graf renklendirme, graf üzerinde birbirine komşu olan düğümlere farklı renk atama işlemidir; amaç, en az sayıda renk kullanılarak tüm düğümlere komşularından farklı birer renk vermektir. Renklendirmede kullanılan toplam renk sayısı kromatik (chromatik) sayı olarak adlandırılır.

Uygulamada, graf renklendirmenin kullanılacağı alanların başında, ilk akla gelen, harita üzerindeki bölgelerin renklendirilmesi olmasına karşın, graf renklendirme bilgisayar biliminde ve günlük yaşamdaki birçok problemin çözümüne ciddi bir yaklaşımdır.

### Graf Renklendirme Problemi İçin Yazdığım Algoritma Ve Çalışma Zamanı Analizi

Öncelikle analizi yapmadan önce şunu belirtmek isterim. Algoritma analizi yapılırken araştırmalarıma göre kod üzerinden bir çalışma zamanı analizi yapmak (Javada `System.nanoTime();`) çok sağlıklı ve kullanışlı değil. Bunun sebebi bir algoritma çalışırken, çalıştığı bilgisayarın işlemcisi, registrları, ekran kartı gibi bir çok parametreye bağlı değişebilir. Ben ise analiz yaparken kodun içerisine girip matematiksel denklemler üzerinden analiz yapmayı tercih ettim böylelikle herhangi bir bilgisayarın işlemci gibi parametrelerine bağlı kalmadan daha sağlıklı bir analiz yapabileceğimize inanıyorum. Fakat ödevin çok daha geniş bilgi oluşturması için grafiklerde bunu da ekleyeceğim.



```
1 public class grapRenklendirmeÇalışmaZamanı {
2     final int köşeSayısı = 4; //----->1
3     int[] renkler; //----->1
4     //+2
5
6     boolean renkAtamaKontrolü(int köşeSayısı, int grap[][] , int renk[],
7         int c)
8     {
9         for (int i = 0; i < köşeSayısı; i++){ //----->köşeSayısı
10             if (grap[köşeSayısı][i] == 1 && c == renk[i]){ //----->köşeSayısı
11                 return false; //----->köşeSayısı
12             }
13         }
14         return true; //----->1
15     } //3*köşeSayısı+1
16
17     boolean grapKontrolü(int grap[][] , int renkSayısı,
18         int renkler[], int köşe)
19     {
20         if (köşe == köşeSayısı) //----->1
21             return true; //----->1
22
23         for (int c = 1; c <= renkSayısı; c++) //----->renkSayısı
24         {
25             if (renkAtamaKontrolü(köşe, grap, renkler, c)) //----->renkSayısı
26             {
27                 renkler[köşe] = c; //----->renkSayısı
28
29                 if (grapKontrolü(grap, renkSayısı, renkler, köşe + 1)) //----->renkSayısı
30                     return true; //----->renkSayısı
31                 renkler[köşe] = 0; //----->renkSayısı
32             }
33         }
34         return false; //----->renkSayısı
35     }
36 }
```

Bu yazdığım algoritmanın renkAtamaKontrolü ve grapKontrolü metodlarıdır. Burada renkAtamaKontrolü metodu köşe sayısı ve renk sayısının uyuşup uyuşmadığını kontrol ederek boolean tipinde değer gönderir. Bu metodun karmaşıklığı fotoğrafta görüldüğü gibi  $(3 \cdot köşeSayısı + 1)$ 'dir.

Bir diğer metod olan grapKontrolü de bir genel kontrol metodudur. Kısaca anlatmak gerekirse main metodunda tanımladığımız matrise göre ,renk sayısına,renklere ve köşe sayısına göre diğer metodlara gitmemizi sağlar. Bu metodun karmaşıklığı ise  $(6*\text{renkSayısı}+3)$ 'dür.

```
        renkler[köşe] = 0; //----->renkSayısı
    }

    return false; //----->1
} //----->6*renkSayısı+3

boolean grapRenklendirme(int grap[][], int renkSayısı)
{
    renkler = new int[köşeSayısı]; //----->1
    for (int i = 0; i < köşeSayısı; i++) //----->köşeSayısı
        renkler[i] = 0; //----->köşeSayısı

    if (!grapKontrolü(grap, renkSayısı, renkler, 0)) //----->1
    {
        System.out.println("Renklendirme yapılamadı"); //----->1
        return false; //----->1
    }

    ekranaÇıktıyıYazdırma(renkler); //----->1
    return true; //----->1
} //----->2*köşeSayısı+6

void ekranaÇıktıyıYazdırma(int renkler[])
{
    System.out.println("Renklendirme yapıldı : Renkler sırasıyla aşağıdadır."); //----->1
    for (int i = 0; i < köşeSayısı; i++) //----->köşeSayısı
        System.out.print(" " + renkler[i] + " "); //----->köşeSayısı
    System.out.println(); //----->1
} //----->2*köşeSayısı+2
```

Bu sayfa ise grapRenklendirme ve ekranaÇıktıYazdırma olmak üzere iki metoddan oluşmaktadır. Birinci metod olan grapRenklendirme graftaki köşelerin herbirine komşuların aynı olmamak üzere bir renk atar. Aynı zamanda karmaşıklığı da  $(2*\text{köşeSayısı}+6)$ 'dır

İkinci metod olan ekranaÇıktıYazdırma sırasıyla ekrana köşelerin renklendirmesini renklerin dağılımını yazar. Karmaşıklığı ise  $(2*\text{köşeSayısı}+2)$ 'dir.

The screenshot shows an IDE with the following components:

- Projects:** A tree view on the left showing a project named 'graphColoring' with sub-packages like 'Source Packages', 'Test Packages', and 'Libraries'. It also shows a 'Huffman' project.
- Source:** The main editor window displaying the code for 'grapRenklendirmeÇalışmaZamanı.java'. The code includes a main method that creates a 'Coloring' object, prints a graph, and calls 'Coloring.grapRenklendirme'.
- Output - graphColoring (run):** The bottom panel showing the execution output. It displays the graph structure, the time taken for coloring, and a success message.

```
public static void main(String args[])
{
    grapRenklendirmeÇalışmaZamanı Coloring = new grapRenklendirmeÇalışmaZamanı(); //----->1
    /* (3)---(2)
       | / |   3---2439388085330      24527207066600
       | / |   4---24431601529100
       | / |   24457387861400
       (0)---(1) 24561589543300
    */
    int grap[][] = {
        {0, 1, 1, 1},
        {1, 0, 1, 0},
        {1, 1, 0, 1},
        {1, 0, 1, 0},
    };
    //----->1
    int renkSayısı = 50; //----->1
    Coloring.grapRenklendirme(grap, renkSayısı); //----->1
    System.out.println("Geçen süre : "+System.nanoTime());
} //----->4
//Bu algoritmanın yürütme zamanı =7*köşeSayısı+6*renkSayısı+19
//HAZIRLAYAN: ÖMER CENGİZ 16260056
```

Output - graphColoring (run)

```
run:
Renklendirme yapıldı : Renkler sırasıyla aşağıdadır.
1 2 3 2
Geçen süre : 25585145535300
BUILD SUCCESSFUL (total time: 0 seconds)
```

Son fotoğrafta da kodun main kısmını ve outputu görüyoruz. Main kısmı ise sınıfımızdan oluşturduğumuz Coloring nesnesiyle grapRenklendirme metodunu ve ona bağlı olan yan metodları çalıştırıyor. Main metodunun karmaşıklığı ise 5 tir.

Böylelikle her metodun ve metodların dışındaki global değişkenlerinde çalışma zamanlarını topladığımızda **(7\*köşeSayısı+6\*renkSayısı+19) olan çalışma zamanı denklemini bulmuş oluyoruz**. Geriye ise bu algoritmanın iyi, kötü ve ortalama çalışma zamanlarını hesaplayarak grafiğe dökmek kalıyor.

Grafik çizimine geçmeden önce şunu belirtmek gerekiyor. T(n) denkleminiz köşeSayısı ve renkSayısı olmak üzere iki değişkene bağlı olduğundan ben burda renk sayısını sabit tutucam (Bir düğümün komşu düğümlerinden rengi farklı olmalı kuralını bozmadan).

### En İyi Durumda Çalışma Zamanı Hesabı:

$$T(\text{köşeSayısı}, \text{renkSayısı}) = 7 * \text{köşeSayısı} + 6 * \text{renkSayısı} + 19$$

$$\text{renkSayısı} = 3$$

köşeSayısı = 1 (köşe sayısının 1 köşeSayısı kadar dönen döngülerin 1 kere çalışması anlamına gelecek böylelikle renklendirme yapamayacak bir çok işlem gerçekleşmeyecektir. Bu da en iyi durumu hesaplamak için istenilen şeydir.)

$$T(\text{köşeSayısı}, \text{renkSayısı}) = 7 * 1 + 6 * 3 + 19 = 44$$

### Ortalama Durumda Çalışma Zamanı Hesabı:

$$T(\text{köşeSayısı}, \text{renksayısı}) = 7 * \text{köşeSayısı} + 6 * \text{renksayısı} + 19$$

$$\text{renkSayısı} = 3$$

$\text{köşeSayısı} = 3$  ( $\text{köşeSayısı}$ nı 3 yapmamın sebebi ortalama durumu daha iyi göstermek istemem buna bağlı olarakta kurala uymak için renk sayısını 3 yaptım)

$$T(\text{köşeSayısı}, \text{renksayısı}) = 7 * 3 + 6 * 3 + 19 = 58$$

### Kötü Durumda Çalışma Zamanı Hesabı:

$$T(\text{köşeSayısı}, \text{renksayısı}) = 7 * \text{köşeSayısı} + 6 * \text{renksayısı} + 19$$

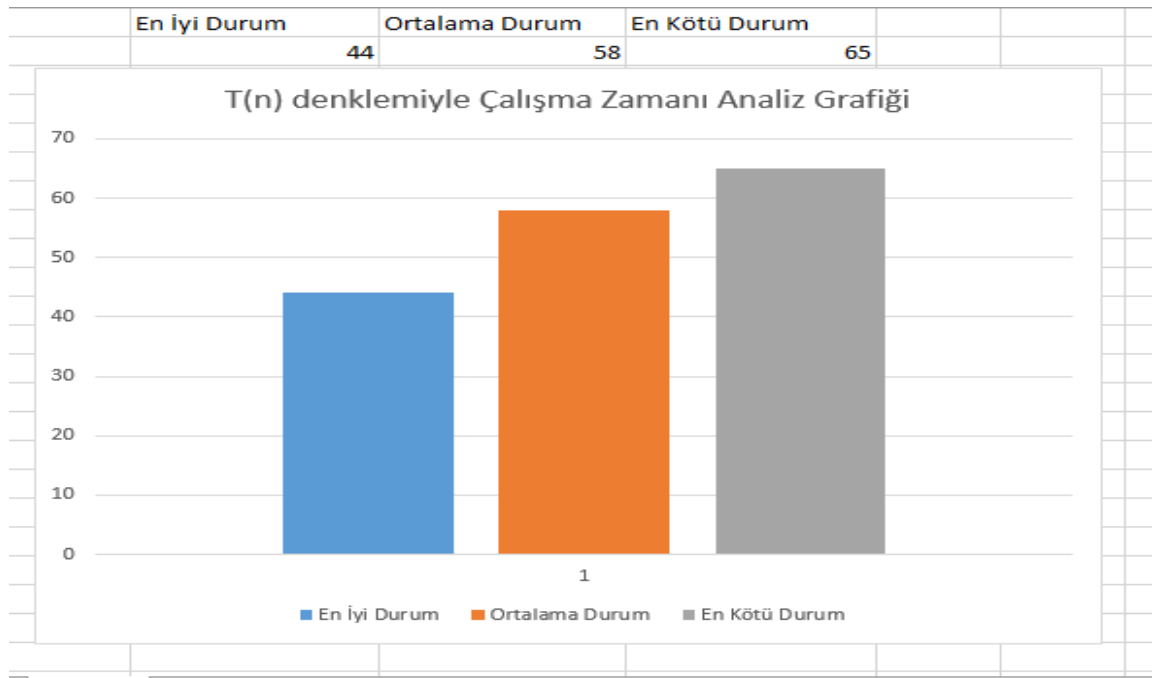
$$\text{renkSayısı} = 3$$

$\text{köşeSayısı} = 4$  ( $\text{köşeSayısı}$ nı 4 yapmamın sebebi en iyi durumu daha iyi göstermek istemem ve renklendirme kuralına göre 3 renk için en fazla 4 köşesi olmalı buna bağlı olarakta kurala uymak için renk sayısını 3 yaptım)

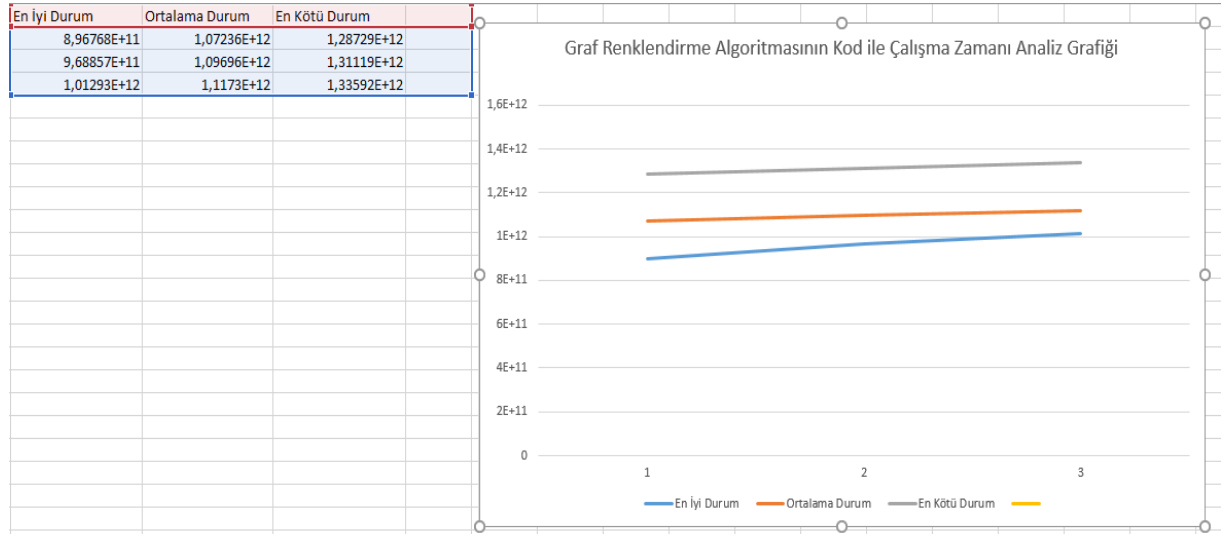
$$T(\text{köşeSayısı}, \text{renksayısı}) = 7 * 4 + 6 * 3 + 19 = 65$$

$$\text{Karmaşıklık} = O(\text{köşeSayısı})$$

### T(n) denklemiyle yapılan çalışma zamanı analizi grafiği:

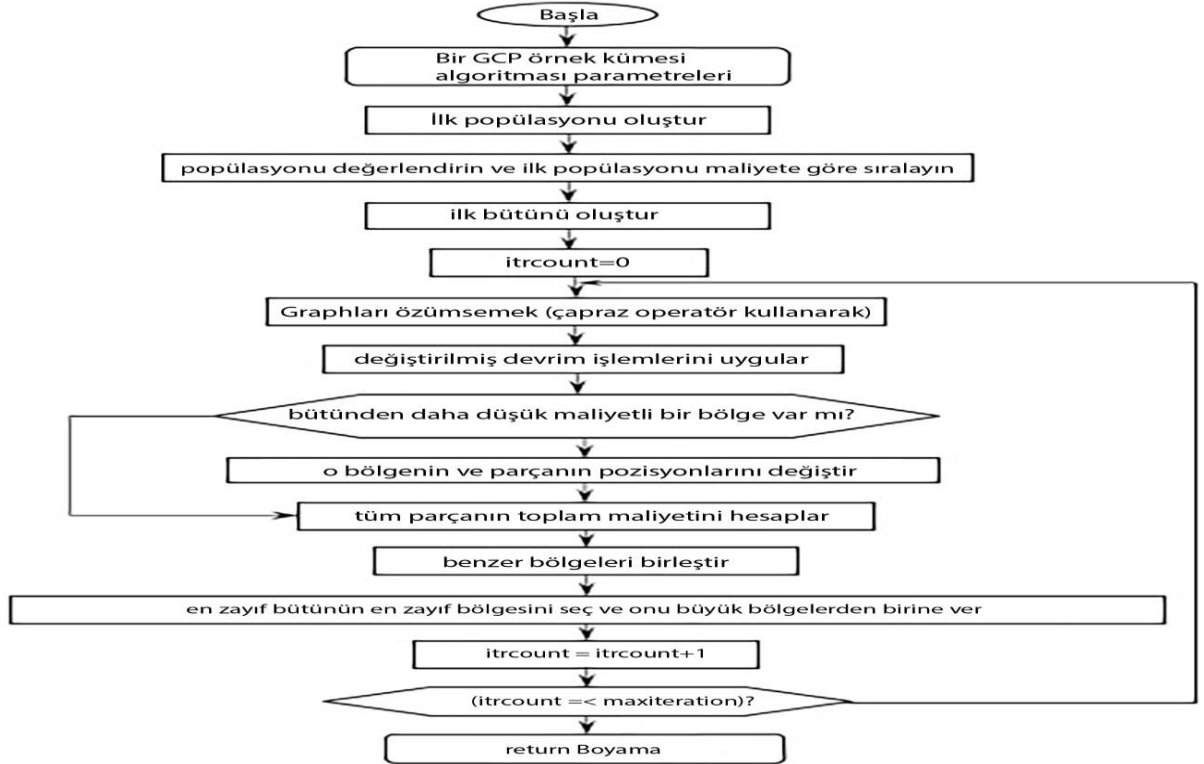


## Kodlama ile yapılan çalışma zamanı analizi grafiği (Java'da System.nanoTime()):



Değerlerin hepsi nano saniye cinsindendir.

## AKIŞ DİYAGRAMI



HAZIRLAYAN : ÖMER CENGİZ 16260056