T.C. Fırat Üniversitesi
Bilgisayar Mühendisliği Bölümü

BMÜ316-Algoritma Analizi Dersi

Algoritma ve Karmaşıklık Ödevi Raporu

Ders Sorumlusu: Prof. Dr. Mehmet KARAKÖSE

Öğrenci İsim: Mert İNCİDELEN
Öğrenci No.: 170260101

Ödev Tanımı

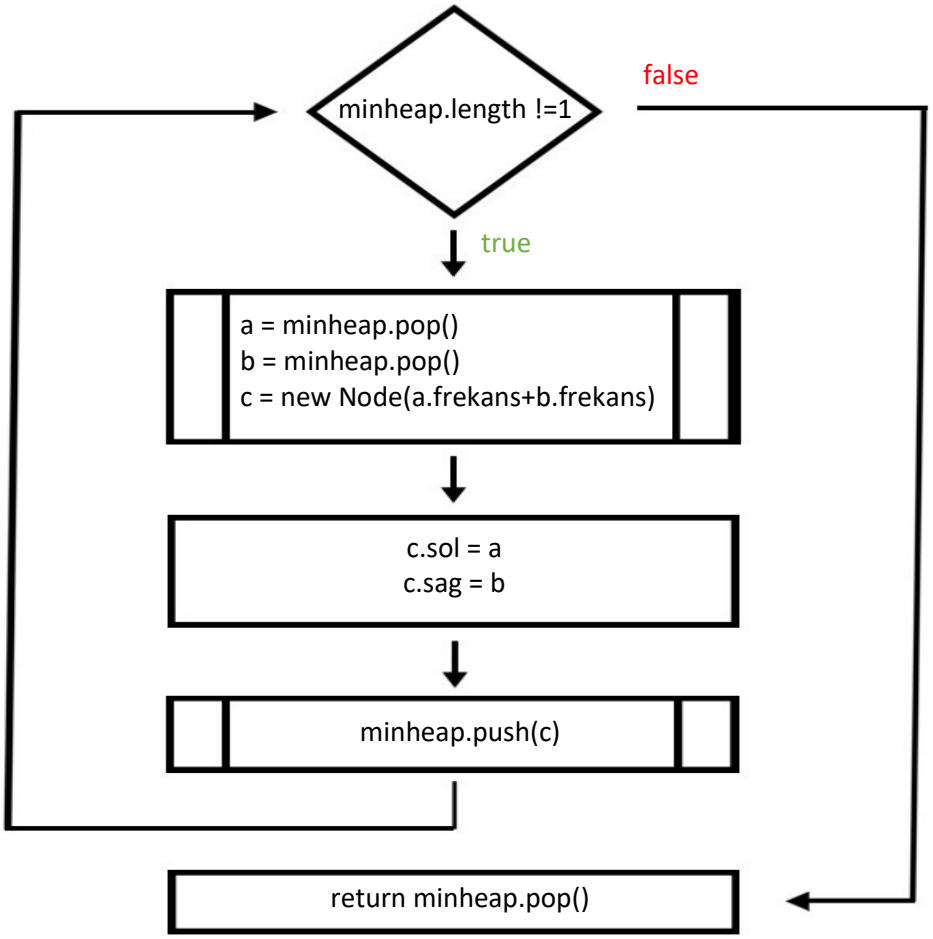
BM9 316 - Algoritma Analizi Dersi 2. Ödevinde,
1. Verilen konulardan en az iki tanesini seçerek algoritmaların karmaşıklığının bulunması, akış şemasının ve zaman tüketim tablosu ile grafiğinin verilmesi;
2. Cyclomatic Complexity kavramının açıklanması, niçin kullanıldığının ve nasıl uygulandığının en az iki örnek ile gösterilmesi beklenmektedir.

1. Huffman Encode ve Decode Algoritmaları

Huffman Algoritması, bir metinde tekrar eden karakterleri tekrar etme sayılarına göre ikili ağaca yerleştirerek sıkıştırma sağlamaktadır.

- **ENCODE**

Huffman Algoritması, kodlama stratejisi şöyle işlemektedir:
1.ADIM: Her karakter için bir düğüm oluşturulur. Düğümde karakter ile birlikte, karakterin tekrar etme sayısı, sağ ve sol düğümler tutulur.
2. ADIM: Bu düğümler karakterlerin tekrar etme sayılarına göre minheap'e atılır.
3. ADIM: Yığındaki en az tekrar eden iki karakterin düğümü listeden çıkarılır.
4. ADIM: Çıkarılan düğümlerin karakter tekrar etme sayıları toplamlarından bir düğüm oluşturulur ve bu düğüme çıkarılan iki düğümden frekansı düşük olan sol çocuk büyük olan sağ çocuk olacak şekilde düğümler eklenir.
5. ADIM: Yeni düğüm daha sonra sıralı listeye eklenir.
Listede tek düğüm kalana kadar 3-5 adımları devam eder. Listede kalan tek düğüm Huffman Ağacının kök düğümü olmaktadır.
Algoritmanın akış şeması şu şekilde gösterilebilir:

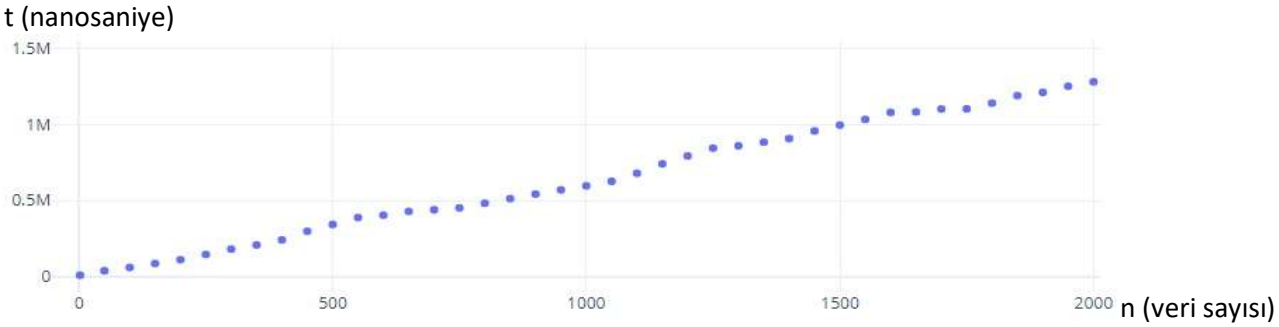


Zaman Karmaşıklığı: O(nlogn)

Her ağacın ağırlığını depolamak için minheap yapısı kullanılır, her işlemde yeni düğümü eklemek **O(logn)** süresi gerektirir. Bu işlem n eleman için **n kez tekrarlanmakta** ve dolayısıyla zaman karmaşıklığı **O(nlogn)** olmaktadır.

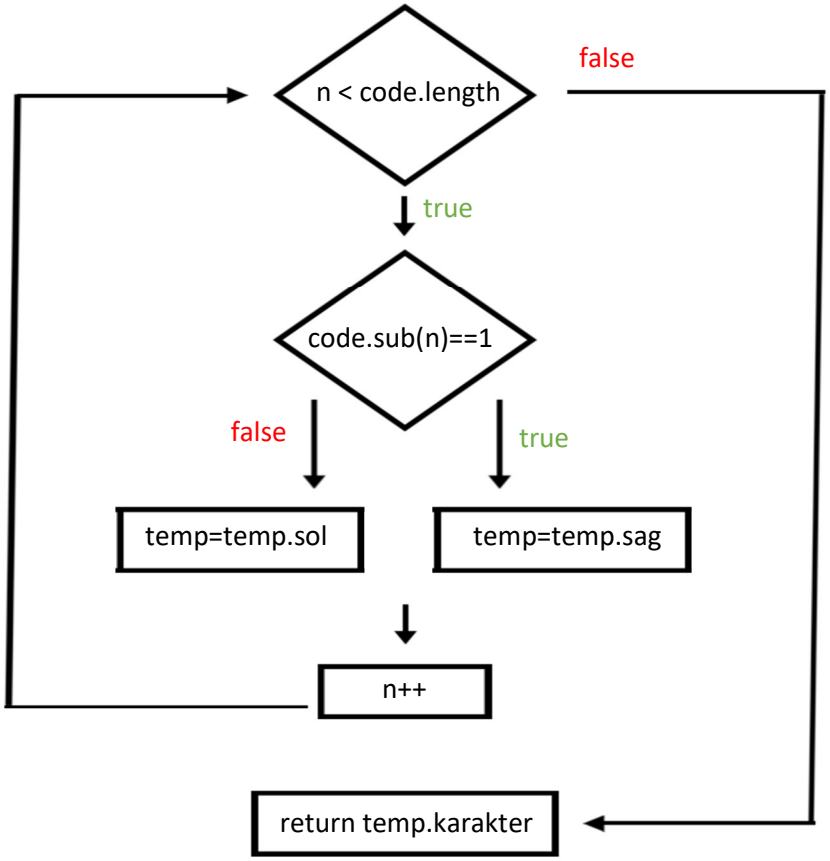
Algoritma Java dilinde bilgisayar üzerinde yürütülmüş rastgele karakter frekans değerleri (n) miktarı arttırılarak huffman yerleştirme sürecinde ortalama zaman tüketimi analiz edilmiştir. Buna göre aşağıdaki tablodaki sonuçlara ve grafikte yer alan verilere ulaşılmıştır.

n (veri)	2	250	500	750	1000	1250	1500	1750	2000
t (ns)	10600	116800	344000	462600	598800	805000	996400	1103300	1281100



- **DECODE**

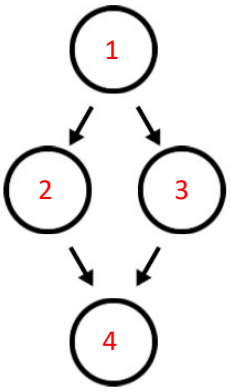
Huffman Algoritması, kod çözme stratejisi ise şöyle işlemektedir:
1.ADIM: Kodun bit değerleri ayrılır. Kökten başlanarak ilerlenir.
2.ADIM: Bit değeri 1 ise sağ çocuğa, 0 ise sol çocuğa ilerlenir.
3.ADIM: Ayrılan bit değerlerinin tümü için 2. adım tekrar edilir. Varılan düğümdeki karakter verisi, kod için çözümlenmiş karakterdir.
Algoritmanın akış şeması şu şekilde gösterilebilir:



2. Cyclomatic Complexity

Cyclomatic Complexity, geliştirilen programda yazılan kodların karmaşıklığının analizi için kullanılan bir karmaşıklık ölçütüdür. Cyclomatic Complexity, programın kontrol akış grafiği çıkarılarak analiz edilebilir. Akış grafiğinde her düğüm komutlara karşılık gelir ve sonraki komut bu komuttan hemen sonra icra edilecek ise komutların işleyiş yönünü gösteren bir kenar ile düğümler birbirilerine bağlanır.
Buna göre aşağıdaki örnek kod parçası için akış grafiği şekilde gösterildiği gibi olmalıdır.

```
[1]   if  n < 50 then
[2]         print "Sayı 50'den küçük"
      else
[3]         print "Sayı 50'ye eşit veya büyük"
[4]   end if
```

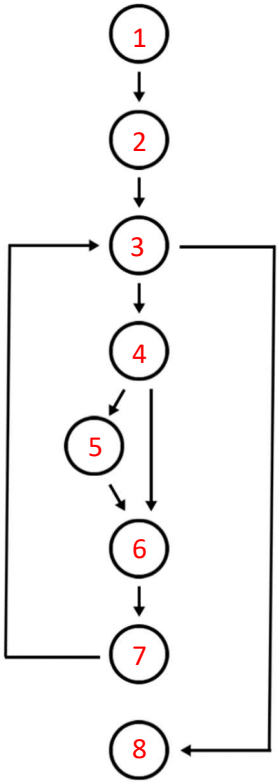


Karmaşıklık, **E – N + 2P** olarak hesaplanır. Buradaki E, N ve P değişkenleri;

- **E** kontrol akış grafiğindeki kenar sayısını,
- **N** kontrol akış grafiğindeki düğüm sayısını,
- **P** bağlı bileşenlerin sayısını ifade etmektedir.

Aşağıda verilen n sayısının pozitif tam bölen sayısını hesaplayan bir kod parçası yer almaktadır. Bu kodun kontrol akış grafiği şekilde görüldüğü gibi olmaktadır.

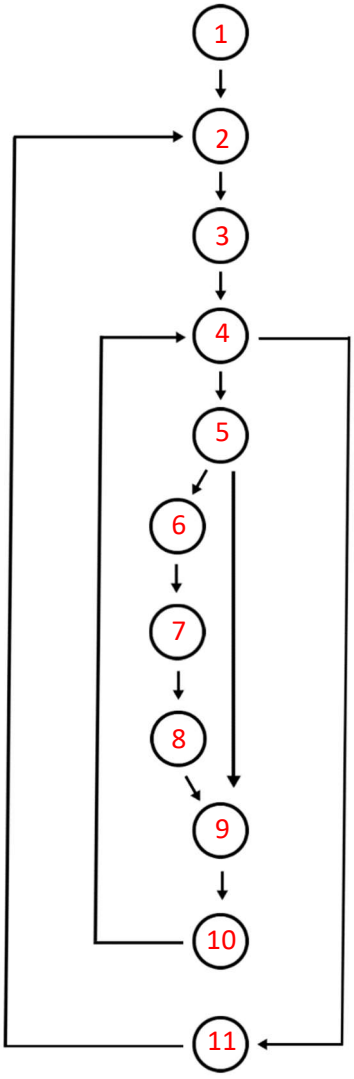
```
      pozitifTamBolenler(n)
[1]   i=1
[2]   sayac=1
[3]   while i < n do
[4]         if n % i == 0 then
[5]               sayac++
[6]         end if
[7]         i++;
      end while
[8]   return sayac
```



Buna göre verilen kodun kontrol akış grafiği üzerinde yer alan düğüm sayısı N=8, kenar sayısı E=9, tek bir yordam bulunduğundan P=1 olarak bulunur. Buna göre verilen kod için Cyclomatic Complexity hesaplaması yapılarak 9 – 8 + 2 = 3 değeri bulunur.

Aşağıda kabarcık sıralama algoritmasına ait bir kod parçası yer almaktadır. Bu kodun kontrol akış grafiği şekilde görüldüğü gibi olmaktadır.

```
kabarciksiralama(dizi[], diziUzunluk)
[1]     i = 0
[2]     while i < diziUzunluk do
[3]         j = 0
[4]         while j < diziUzunluk do
[5]             if dizi[j] > dizi[j + 1] then
[6]                 temp = dizi[j]
[7]                 dizi[j] = dizi[j + 1]
[8]                 dizi[j + 1] = temp
[9]             end if
[10]            j++
            end while
[11]        i++
        end while
```



Buna göre verilen kodun kontrol akış grafiği üzerinde yer alan düğüm sayısı N=11, kenar sayısı E=13, tek bir yordam bulunduğundan P=1 olarak bulunur. Buna göre verilen kod için Cyclomatic Complexity hesaplaması yapılarak 13 – 11 + 2 = 4 değeri bulunur.

```java
//HUFFMAN BILGISAYAR ANALIZINDE KULLANILAN KOD

package HuffmanAlgoritmasi;
import java.util.PriorityQueue;
import java.util.Comparator;

class Node {

    int frekans;
    char karakter;
    Node sol;
    Node sag;
}

class MyComparator implements Comparator<Node> {

    public int compare(Node x, Node y) {

        return x.frekans - y.frekans;
    }
}

public class HuffmanAlgoritmasi {

    public static void main(String[] args) {

    int n=29
        for (int j = 2; j < 29; j++) {

            int n = j;
            char[] karakter = {'a', 'b', 'c', 'ç', 'd', 'e', 'f', 'g', 'ğ', 'h',
            'ı', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'ö', 'p', 'r', 's', 'ş', 't',
            'u', 'ü', 'v', 'y', 'z');
            int[] frekans =
            {90,45,71,32,4,5,78,41,65,79,21,52,3,17,52,69,75,41,32,85,94,63,21,75,78,4
            1,23,65,74}

            PriorityQueue<Node> kuyruk
                    = new PriorityQueue<Node>(n, new MyComparator());

            long baslamaZamani = System.nanoTime();
            for (int i = 0; i < n; i++) {
                Node dugum = new Node();

                dugum.karakter = karakter[i];
                dugum.frekans = frekans[i];

                dugum.sol = null;
                dugum.sag = null;
                kuyruk.add(dugum);
            }

            Node root = null;
            while (kuyruk.size() > 1) {
                Node a = kuyruk.peek();
                kuyruk.poll();
                Node b = kuyruk.peek();
                kuyruk.poll();
                Node c = new Node();
                c.frekans = a.frekans + b.frekans;
                a.karakter = '*';
                c.sol = a;
                c.sag = b;
                root = c;
                kuyruk.add(c);
            }
//
////        kod çözme:
//        String code="000";
//        Node temp=root;
//        for (int i = 0; i < code.length(); i++) {
//            String bit=code.charAt(i)+"";
//
```

```
70  //              if("1".equals(bit)){
71  //                  temp=temp.sag;
72  //              }
73  //              else{
74  //                  temp=temp.sol;
75  //              }
76  //          }
77  //          System.out.println(temp.karakter);
78
79                long bitisZamani = System.nanoTime();
80                long gecenZaman = bitisZamani - baslamaZamani;
81                System.out.print("Veri adedi: " + n + " ");
82                System.out.println(gecenZaman + " nanosaniye");
83            }
84        }
85    }
```