```
#bst deki yaprakları silen kod
def yaprak_sil(self):
    if not hashchild(self.left):
        self.left = None
    else:
        yaprak_sil(self.left)

    if not hashchild(self.right):
        self.right = None
    else:
        yaprak_sil(self.right)

#BST'deki tüm değerler sayılar iken, ORTALAMA(self) hesaplayan yöntemi gerçekleyin
def topla(self):
    if self == None:
        return 0
    else:
        return self.key +topla(self.right)+topla(self.left)

def say(self):
    if not self:
        return 0
    else:
        return 1 + say(self.left)+ say(self.right)

def ortalama(self):
    return topla(self)/say(self)
```

```
#parametre olarak gönderilen sayıdan küçük elemanları ekrana yazdıran YAZDIR metodunu yazınız.
def yazdir(bst,esik):
    if bst.kok < esik:
        print bst.kok
    if bst.sag:
        yazdir(bst.sag)
    if bst.sol:
        yazdir(bst.sol)
#hashtabloya ekleme yapacak ve çakışmada zincirleme kullanılacak kod
def Ekle(htablo,dizgi):
    h = hash(dizgi)
    if htablo[h] == None:
        htablo[h] = [dizgi]
    else:
        htablo[h].append(dizgi)

#hash ortalama
def Ortalama(htree):
    sz = len(htree)/2
    for in range(1,sz+1):
        print htree + "nin ortalamsı"
        tmp = htree[2*i]
        if(2*i+1) <=sz:
            tmp += htree[2*i+1]
        tmp/=2
        print tmp

#iki düğüm arasında yol (kenar değil) olup olmadığını ölçen
def yolvar(graf,dugum1,dugum2):
    yol,s = [],[dugum1]
```

```
    while len(s):
        d = s.pop(0)
        yol.append(d)
        if d == dugum2:
            return True
        for e in graf[d]:
            if (not e in yol) and (not e in s):
                s.push(0,e)
    return False

#Bu grafın kenar sayısını hesaplayan KENARSAY(graf) metodunu yazınız
def KenarSay(graf):
    kenar = {}
    for d1 in graf.keys():
        for d2 in graf[d1]:
            a = min(d1,d2)
            b = max(d1,d2)
            kenar[(a.b)] = 1

    return len(kenar.key())

#Maksimum düğüm derecesinin kaç olduğunu bulan
def maxdugumderece(graf):
    derece = []
    for d in graf.key():
        derece.append(len(graf[d]))
    return max(derece)
```