DÖNÜŞTÜR YÖNET (Transform & Conquer)

Tasarım daha kolay hale gelir
Aynı problemin farklı ifade ediliş biçimi
Farklı ifade ediliş biçimine daha önce çözüm bulunmuş

En çok kullanılan Teknik

PreSorting (Önce Sırala)
Nerelerde kullanılıyor?

1. Arama yapılacaksa
2. Orta değerin bulunması
3. Tek olan elemanın bulunması (unique)
4. Birçok geometrik problemde
    a. Noktaların koordinatların sıralanması

PreSorting Kullanarak Arama
k değeri A[0-100] aranıyor

- En etkili bir algoritma ile A dizisini sırala
    - MergeSort *kullandığımız varsayalım* En kötü durumda $O(n.logn)$
- İkili aramayı kullan $O(logn)$
- **PERFORMANS :** $O(nlogn)+O(logn)=O(nlogn)$

Presortin Kullanarak Tek olan elemanın bulunması
Bir kere yapmak yetmiyor, Hepsini diğerleri ile karşılaştırmak gerek

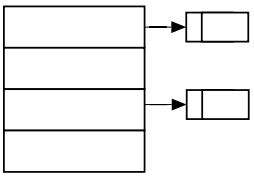
- Etkili bir algoritma ile birdiziyi sırala
    - MergeSort O(nlogn)
- Dizide eleman eleman ilerle ve komşulara bak
    - O(n)
- **PERFORMANS :** O(nlogn)+O(n)=O(nlogn)

Brute Force – Kaba Kuvvet Kullansaydık

- İç içe for döngüsü
    - $O(n^2)$ // Bütün elemanları diğer bütün elemanlar ile karşılaştır.

Daha iyileştiremez miyiz?

HashCode



- En kötü durum O(n)
- Tüm elemanların aynı indisle bulunması

Dizide en sık geçen elemanı bulmak
BruteForce kullanılmış olsa

$$C(n) = (n-1) + (n-2) + (n-3) \cdots + 1$$

$$C(n) = \sum_{i=1}^{n} i - 1 = \frac{(n-1) \cdot n}{2}$$

$$O(n^2)$$

PreSort (Önce Sırala) Kullanılsaydı

rl : O anki seçilen elemanın kaç adet olacağı
rv : O anki seçilen elemanın ne olduğu

PresortMode[$A[0 \cdots n-1]$ ***]***

```
A Dizisini Sırala
i = 0;   mf = 0;
while(i < n)
{
rl = 1;   rv = A[i];
          while(i + rl < n   &&   A[i + rl] == rv)
          {
          rl = rl + 1;
          }
          if (rl < mf)
          {
                    mf = rl;
                    mv = rv;
          }
          i+ = rl;
}         // endwhile
```

mf En yüksek frekansa sahip eleman

n elemanlı dizi

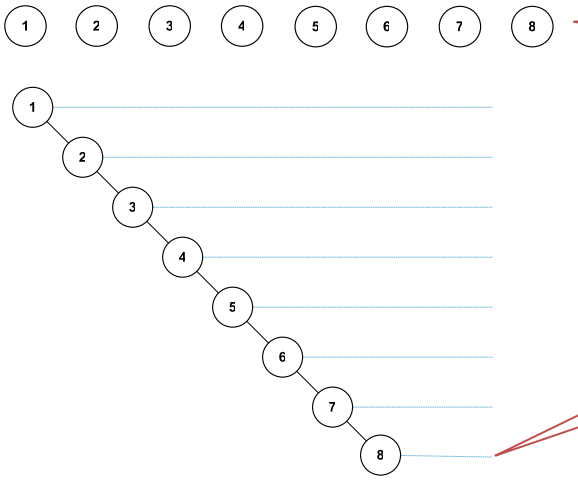
İç içe iki döngü gibi görülse de daha az çalışır

Analiz

a. Sıralama için $O(\log n)$ // MergeSort
b. Dıştaki while } $O(n)$ // En iyi ve En kötü durum için
c. İçteki while }

Maliyet $O(n) + O(\log n) = O(\log n)$

Ağaçlar üzerinde PreSort Kullanımı
İkili Arama Ağacı



En kötü durum sıralı olması
Karma dizide eleman aramaya benzer

Yükseklik arttıkça performans düşer
Performans ağacın yüksekliği ile ilgilidir.
$h =$ Yükseklik $\log n \le h < n$

En Kötü Durum	$O(n)$
Ortalama Durum	$O(\log n)$
En iyi Durum	$O(1)$

Her iki durumda da amaç logn'e yaklaştırmak

1. AVL Ağaçları
2. RBT (Kırmızı Siyah Ağaçlar)

AVL Ağaçları **<u>yüksekliği</u>** baz alır.
Yükseklik;
Çocuğu olmayan kök = *0*
Düğümü olmayan ağacın yüksekliği = *-1*

Bir ağacın <u>derinliği</u> olamaz. // Anlamsız

En uzaktaki yaprağa olan uzaklık
$Yükseklik = 1 + Max(sol, sağ)$

AVL Ağaçları
Denge Bozulduğu an dengeyi sağlar

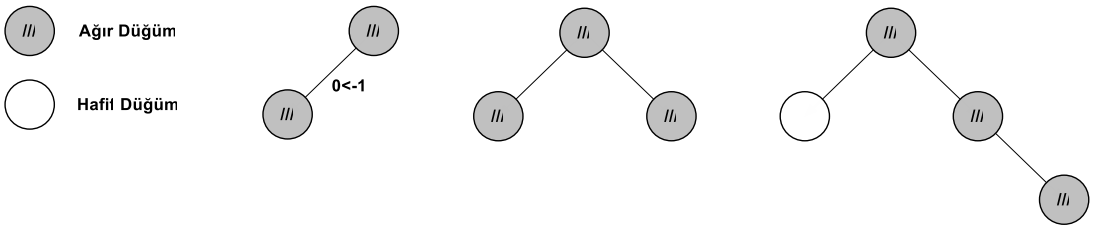
Bu iki koşulu sağlayan her ağaç **AVL Ağacı**dır

1. Yaprak ve kök olmayan her düğüm mutlaka bir kardeşi olmalıdır.
2. Kardeşlerin yükseklikleri eşit veya en fazla bir (1) fark olmalıdır.

Dengesiz bir ağaç; hafif yada ağır diye nitelendirilebilir.
Kök Ağırdır *// Bir kardeşi olmayacağı için ağırdır*
Hafif Düğüm *// Kardeşin yüksekliği daha fazla ise düğüm hafiftir.*

Hafif olmayan her düğüm **Ağırdır**

Bir düğüm **solAğır** ➔ $yükseklik(sol) > yükseklik(sağ)$
Bir düğüm **sağAğır** ➔ $yükseklik(sağ) > yükseklik(sol)$

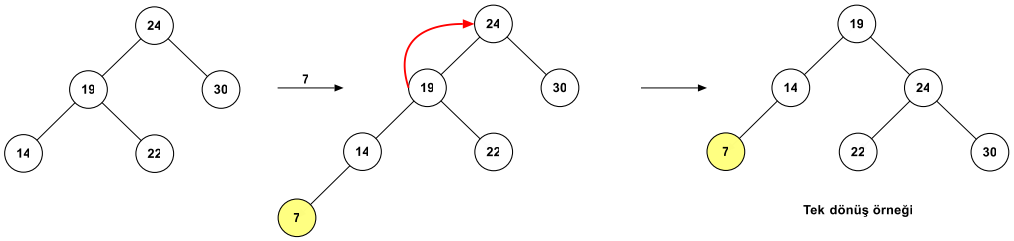


AVL Ağaçlarında;
Tek dönüş veya çift dönüş yapılarak denge sağlanıyor

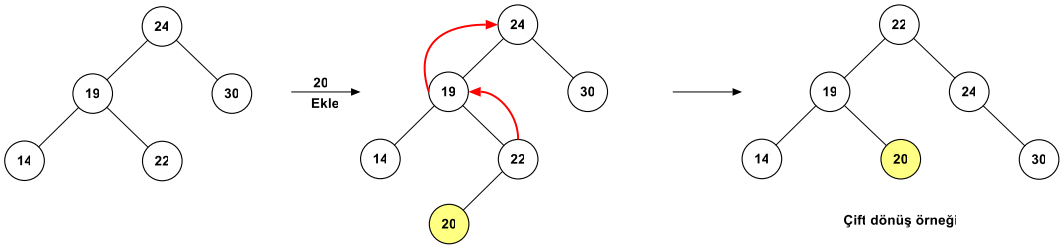
Eğer $yükseklik\left(sol(d)\right) == yükseklik(sağ(d)) + 2$
Eğer $yükseklik \quad sol(sol(d)) == h+1$ ise
Sol çocuk üzerinde tek dönüş yapılacak
Eğer $yükseklik \quad sağ(sol(d)) == h+1$ ise
Sol çocuk üzerinde çift dönüş yapılacak

Eğer $yükseklik\left(sağ(d)\right) == yükseklik(sol(d)) + 2$
Eğer $yükseklik \quad sağ(sağ(d)) == h+1$ ise
Sağ çocuk üzerinde tek dönüş yapılacak
Eğer $yükseklik \quad sol(sağ(d)) == h+1$ ise
Sağ çocuk üzerinde çift dönüş yapılacak

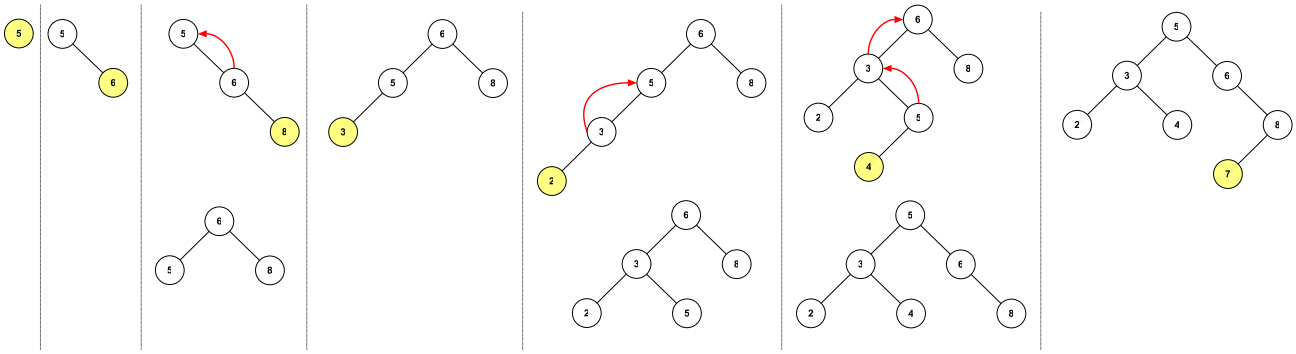
Örnek



Örnek



Örnek
5-6-8-3-2-4-7



AVL Ağacı Analizi

Arama-Ekleme	$O(\log n)$	
Silme	$O(\log n)$	// Çok karmaşık olmasına rağmen
Dengeleme sabiti		// Bir c sabiti kadar yavaşlatır

AVL Ağacı Dezavantajları

- *Dengeleme (Sık Yapılması)*
- *Karmaşıklık*

RBT (Kırmızı Siyah Ağaçlar)

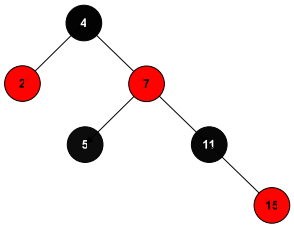
AVL'de eleman silindikten sonraki dengeleme işlemi bizi başka arayışlara yönlendiriyor!

Kırmızı-Siyah Ağalarda güncelleme işlemi $O(1)$ **'i garanti ediyor**

RBT Özellikleri

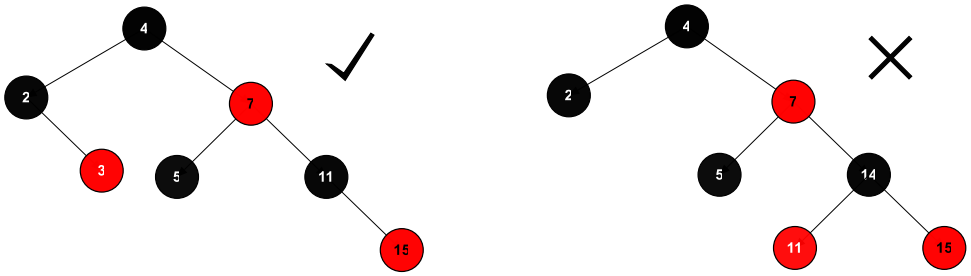
- Kök her zaman siyahtır
- Eğer bir düğüm kırmızı ise çocukları siyah olmalıdır
- Yaprakları köke giden her yolda aynı sayıda siyah düğüm vardır.
- Bu şartları sağlayan ağaç Kırmızı-Siyah Ağaçtır

Örnek



Teorem
k adet anahtara sahiptir
k adet anahtara sahip bir RB ağacının yüksekliği $h \le 2\log(k+1)$ *h=Yükseklik*

Ekleme işlemi
Yeni düğüm kırmızı olarak eklenir
Eğer kural bozulursa *renk değiştir veya döndür*



Derinlik Farkı ≤ 2 *olduğunda sağlıyor*

Herhangi bir düğüm **arama işlemi**	$O(\log n)$
Bir düğümün eklenmesi	$O(\log n)$
Bir düğümün silinmesi	$O(\log n)$
Dengeleme	$O(1)$