**Ağaç Veri Yapısı**

* İlk defa elektrik devrelerinin çiziminde kullanılmıştır -> graf kullanılır
* Graf düğüm ve keranarlardan oluşur. Düğümler birbirine kenarlarda bağlanır. Tek gönlü veya çift yönlü olabilir

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Ağaçlar döngü içermeyen ve bağlı olan graflardır. Her ağaç bir graftır ama her graf bir ağaç değildir. Yani ağaç olabilmesi için grafların bağlı olması gerekir.

Aşağıdaki bir graftır ama ağaç değildir. Çünkü bağlı değil

metin, küçük resim içeren bir resim

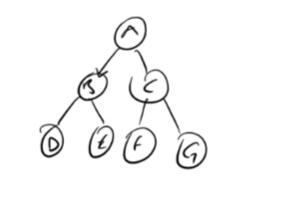
Açıklama otomatik olarak oluşturuldu

Aşağıdaki grafta da döngü olduğundan ağaç değildir.

askı, gerdanlık, makas içeren bir resim

Açıklama otomatik olarak oluşturuldu

Aşağıdaki graf bir ağaçtır



* Ağaçta tüm düğümlere ulaşabileceğimiz bir düğümü hafızada tutmalıyız: kök düğüm -> A
* Sıralama, arama, veri sıkıştırma, veriye erişim, harddisklerin veri okuma prensipleri, veri tabanlarında (örneğin select yazdığımızca tüm işlemler gelir) gibi bilişim dünyasında çok fazla yerde kullanılır
* Derste işleyeceğimiz ağaçlarda bir düğümün iki çocuğu olabilir. Ama daha fazla çocuğu olan ağaçlar da vardır.
* Yaprak düğüm: hiç çocuğu olmayan düğümlerdir -> D E F G
* Kardeş Düğüm: Aynı ebeveyne sahip düğümlerdir -> B-C, D-E, F-G
* Ebeveyn düğüm: ata düğüm -> F -> C, C -> A
* Çocuk düğüm: A’nın çocuğu B ve C dir
* Agaç yüksekliği: kökten en uzaktaki yaprağa olan kenar sayısı. -> 2 (ağaç yüksekliği = kök yüksekliği). C yüksekliği 1 dir
* Tek düğüm varsa yükseklik 0’dır. Agac yoksa yükseklik -1 dir

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Derinlik: bir düğümün derinliği vardır, ağacın derinliği anlamsızdır. Derinlik herhangi bir düğümün köke olan uzaklığıdır. Örneğin b düğümünün derinliği 1 dir. Derinlik ve yükseklik aynı şey değildir. G nin yükseliği 0 ken derinliği 1 dir. A nın yüksekliği 2, derinliği 0 dır. Agaç derinliği anlamsızdır. Kök derinliği 0 dır.

**İfade Agaçları**

İfade ağaçları işlem önceliğine göre tasarlanır. Parantezler ağaca konulmaz.

metin, saat, kol saati içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, saat içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Ağaçları Dolaşmak**

* Dört farklı yol izlenir. İnorder, preorder, postorder, levelorder.
* Ve inorder dolaşttığında oluşan yapıya infix, preorder da prefix, postorder da postfix denir.
* Tüm dolaşmalar kökten başlar. İlk mesaj köke gider
* Agaclarda veri tekrarı olmaz

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Örneğin inorder dolaşıcaksak 10, +, 8, /, 2

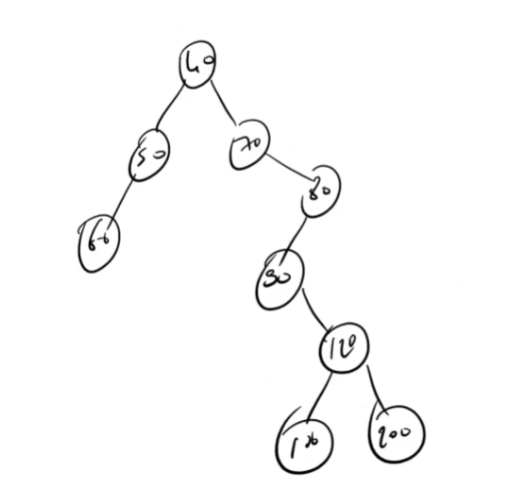
Preorder + 10 / 8 2

Postorder 10 8 2 / +

Level -order da 0. Seviye 1. Seviye 2. Seviye + 10 / 8 2

(burayı anlamadım 1:10 dinle)

Örnek:



İnorder-> köke gel 40, sonra sol çocuk 50 sol çocuk 60 sol çocuk yok 60’ı oku. Şimdi 50 nin kök olduğu ağaçta sol 60’ı okuduk düğümün kendisi 50’yi oku sağ yok. Şimdi en büyük ağaca bak. Solunu okuduk, düğümün kendisi 40’ı oku, şimdi sağa 70. Yine sola git yok, sağa git her kökte sol kendisi sağ yaparak oku

İnorder-> 60 50 40 70 90 100 120 200 80

Preorder-> 40 50 60 70 80 90 120 100 200

Postorder-> 60 50 100 200 120 90 80 70

Levelorder-> (seviyelendir) 40 50 70 60 80 90 120 100 200

**Ağacın Kullanım Yerleri**

**Arama İşlemi**

İkili Arama Ağacı: Amaç bir seri içinde aranan elemanı hızlıca bulmak. (dizi üzerinde ikili arama?)

N elemanlı bir dizice eleman arama O(n)’dir.

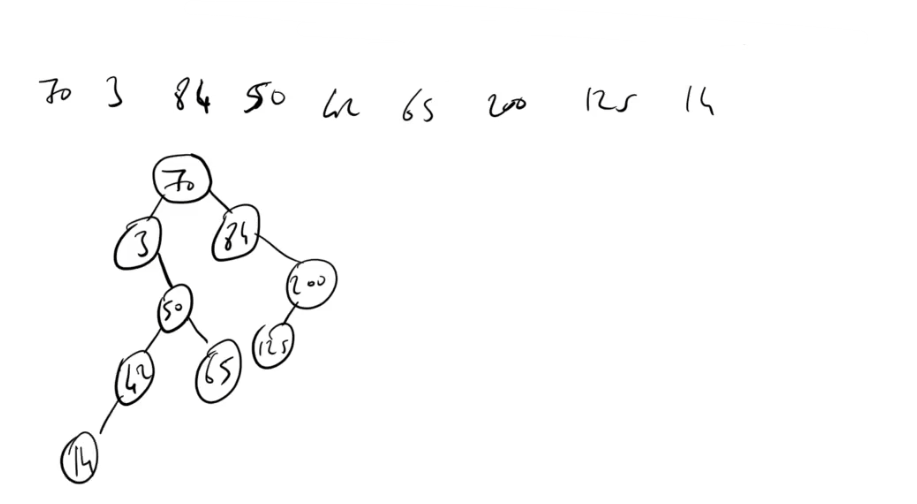
N elemanlı bir listede elaman arama O(n)’dir.

İkili arada ağacında arama O(logn)

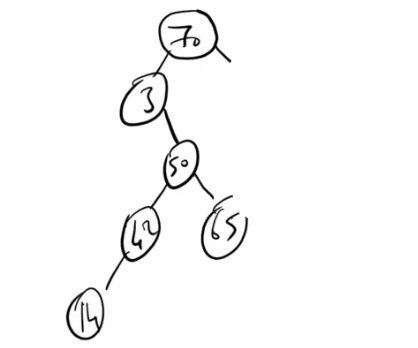
* Bir arama uzayını sürekli yarıya bölerek azaltıyorsan performansı logaritmik düzeydir

Bubble sort O(n^2)

* İkili arama ağacı kuralı: herhangi bir düğüm solundaki bütün düğümlerden büyük, sağındaki bütün düğümlerden küçük olmak zorunda. Eğer değilse ikili arama ağacı doğru çalışamaz



Örneğin 50 var mı diye arayalım. Aramaya kökten başlamamız gerek çünkü elimizde olan adres kökün adresidir. 50, 70’ten büyük mü küçük mü? Küçük o zaman sağ tarafa bakmaya gerek yok. Zaten sağ tarafın 50 den büyük olduğunu biliyoruz.



Şimdi 3 e bakalım. 50 ten büyük o zaman sağa gidicez ve 50 yi bulduk.

* Ağacı yarıya indirdikçe logaritmik düzeye erişiriz

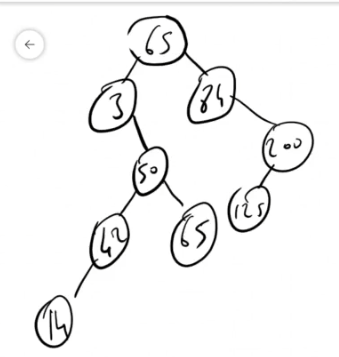
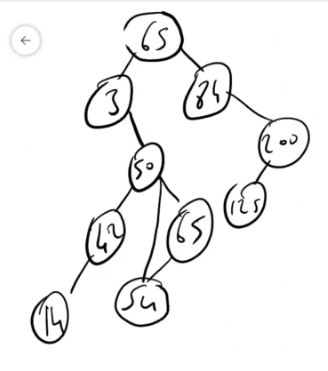
Ayrıca yukarıdaki ağacın yüksekliği 4. (kökün en uzaktaki düğüme (14) olan uzaklığı)

84’ün derinliği 1(köke olan uzaklık), 84’un yüksekliği (en uzak yaprağa olan uzaklık) 2

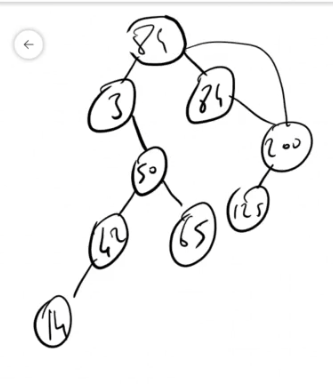
**Silme İşlemi**

Üç kural vardır:

1. Silinecek düğümün hiç çocuğu yoksa direkt düğüm silinir.
2. Tek çocuğu varsa sol ya da sağ farketmez bağıl listede aradaki düğümün çıkarılması gibidir. Silinecek düğümün ebeveyni ile çocuğu bağlanır. Örn 200 çıkacaksa 84 ve 125 bağlanır.
3. İki çocuğu varsa iki yöntem izlenebilir. Sonuç olarak veri yukarı taşınır
   1. Solun en sağı: örneğin 70 i silicez o zaman 65 verisini 70’in yerine yazarız. Düğüm silmeyiz. Sonra en alttaki 65 i silebiliriz. 65 in de çocuğu varsa bir yukarıdakiyle bağlarız sıkıntı diil

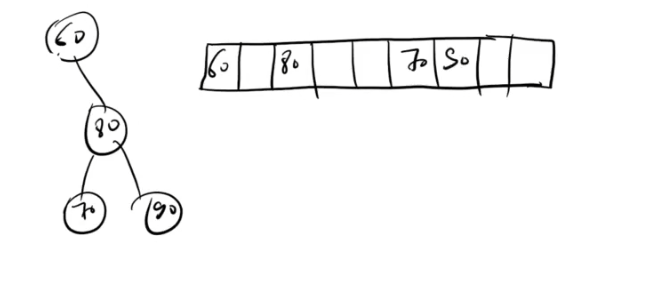
 

* 1. Sağın en solu: en küçük düğümü getiririz yani. 70 yerine 84 yazarız. 84 ü de sileriz



**İkili Arama Ağacının Dizi Üzerinde Gerçekleştirimi (Binary Search Tree)**

Agacı diziye levelorder şeklinde yerleştirirz. Yani çocuk olmasa da onun yerine boşluk bırakılır.



60 yazdık, 60 ın sol çocuğu için boşluk bıraktık sağ çocuğu yazdık sonra 60 ın sol çocuğunun sağ ve sol çocuğu için yer ayırdık sonra 80 inin çocuklarını yazdık. Dizide oluşturmanın dezavantajlarından biri budur. Olmayan düğümlere yer ayırmak zorunda kalırsın.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Sol çocuk: 2 x i + 1

Sağ çocuk: 2 x i + 2

Örneğin 80 in sol çocuğuna gidersek: 80 in indexi 2 2x2+1=5 5. İndexte 70.

* İşte bu yüzden düğüm olsun olmasın dizide yer ayırılır. Çünkü anlık erişim vardır.
* Ebeveyne çıkarken de iki ye bölersin
* Dizide avantaj budur. Anlık erişim vardır. Ebeveyne de direkt çıkabilirsin.
* Listelerde ebevyn adresi tutulmaz çocuklardan ebeveyne direkt erişemezsin. Ancak yukarıdan aşağı inerken ebevbeynin adresini de hafızada tutman gerekir. Ebeveyne erişmen gereken yerlerde dizi üzerinde gerçekleştirme yapman gerekir. Örn sıralama algoritmaları.