İKİLİ AĞAÇLAR VE İKİLİ YIĞINLAR

İkili Ağaçlar

Programlamada çoğu yerde kullanılan veriler üzerinde karar verme işlemi vardır. Verilen bir verinin bir veri kümesi içerisinde olup olmadığını kontrol etmek için karşılaştırmalarla elemanın verilen verinin veriler kümesinde olup olmadığını kontrol etmek amacıyla her karşılaştırma sonucunda iki veri birbirine eşit midir, biri diğerinden büyük mü, küçük mü konusu hakkında karar verilmesi gerekir. Bunun soncunda ağaç şeklinde bir karar mekanizması ortaya çıkar. Bu mantık veri yapısı olarak tanımlanmıştır. Bu şekilde tanımlanan veri yapısına ikili ağaç denir ve ikili ağacın tanımı Tanım 18.1' de görülmektedir.

Tanım 1
Bir ikili ağaç, boş olabilir veya bir kök düğümü ve bu kök düğümünün sağ ve sol alt ağaçlarından oluşur. Sağ veya sol alt ağaçlardan herhangi biri boş olabileceği gibi her ikisi de boş olabilir.

İkili ağaç veri yapısı

```
ağaç=↑düğüm;
düğüm=record
        bilgi:string;
        sağ,
        sol:ağaç;
end;
```

şeklinde tanımlanabilir. Birde ikili ağaçların özel bir şekli olan **ikili arama ağaçları** vardır ve bir sonraki tanımda bu ağaçların özelliği kısaca verilmiştir.

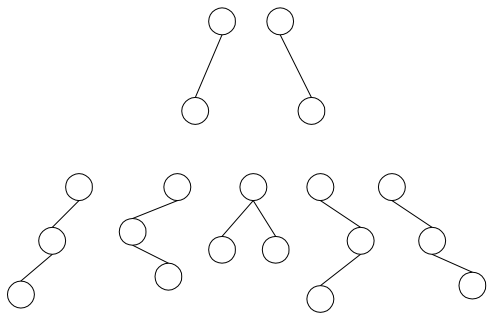
Tanım 2
Bir ikili arama ağacı, boş ağaç olabileceği gibi her düğümünde bir anahtar içeriyor olabilir ve bu anahtarlar aşağıdaki şartları sağlarlar.

- Kök düğümün solunda bulunan bütün düğümlerin anahtarları kök düğümünün anahtarından küçüktür.
- Kök düğümün anahtarı, sağ alt ağacında bulunan bütün anahtarlardan küçük eşittir.
- Kök düğümün, sağ ve sol alt ağaçları da ikili arama ağaçlarıdırlar.

Veri yapısı olarak ikili ağaç ile ikili arama ağaçları arasında hiçbir fark yoktur. Bu iki ağaç arasındaki fark uygulamada ortaya çıkmaktadır. İkili ağaçta düğüm eklemesi için ağacın herhangi bir düğümünde çocuklardan en az biri eksik olmalıdır. Bunun anlamı verinin içermiş olduğu bilginin ağacın oluşumuna hiçbir etkisi yoktur. Fakat bunun tersine ikili arama ağaçlarında ise, verilecek olan veri kümesi ağacın şeklini belirler.

Şekil 18.1' de ikili ağaçlara bazı örnekler verilmiştir. İkili ağaçların veya ikili arama ağaçlarının k gibi bir seviyesindeki maksimum düğüm sayısı 2^k kadar olur ($k \geq 0$).

İkili arama ağaçlarına eleman eklenmesi durumunda; ilk önce eklenecek elemanın anahtar değeri ile ağacın kök düğümünün anahtar değeri karşılaştırılır. Eklenecek elemanın anahtar değeri kök düğümün anahtar değerinden büyükse, sağ tarafa gidilir ve diğer durumda sol tarafa gidilir. Her seviyede aynı işlem tekrarlanır ve NIL değer bulunduğu zaman bu değer yerine bir düğüm eklenir ve eklenecek eleman buraya eklenir. İkili arama ağaçları üzerinde yapılan işlemler üç grupta toplanabilir. Eleman silme, eleman ekleme ve ağaç üzerinde gezinti yapmadır.



Şekil 1. *İki ve üç düğümden oluşabilecek her türlü ikili ağaçlar.*

İkili arama ağaçları üzerinde yapılan gezintiler üç şekilde yapılabilmektedir. Bunlar SıraÖnce (PreOrder), Sıralı (InOrder) ve SıraSonra (PostOrder) şeklinde isimlendirilirler. Sırası ile SıraÖnce algoritması Algoritma 18.1, Sıralı algoritması Algoritma 18.2 ve SıraSonra algoritması Algoritma 18.3' te görülmektedir.

SıraÖnce algoritmasında üzerine gelinen düğümün verisi dışarıya verilir ve ondan sonra sola gidiş varsa, sola gidilir. Sola giden bağlantı NIL ise sağ tarafa gidilmeye çalışılır. Eğer sağ tarafa giden bağlantı da NIL ise o düğümden geriye dönülür.

Sıralı algoritmasında ise ilk önce sola gidilir ve en sonunda NIL değerini gördükten sonra geriye bir adım döner ve üzerinde bulunulan düğümün değeri dışarıya verilir ve ondan sonra sağ tarafı kontrol edilir. Eğer sağ tarafta kayıt varsa, sağa gidilir aksi halde geri dönülür. Bu şekilde ağacın bütün düğümleri üzerinde gezilir.

SıraSonra ağaç gezinmesinde ise, başlangıçta sola gidilmeye çalışılır. Eğer solda bağlantı varsa, o bağlantı takip edilir ve NIL değerine ulaşılınca bir adım geri dönülür ve sağ bağlantı kontrol edilir. Eğer sağ bağlantı varsa, sağa gidilir ve gidilen düğümde tekrar sol bağlantı kontrol edilir. Eğer sol bağlantı varsa, sola gidilir; yoksa sağ kontrol edilir. Bu işleme sağ bağlantı NIL oluncaya kadar devam edilir. Sağ bağlantı NIL olunca bir adım geri dönülür ve o anda üzerinde bulunulan düğümün anahtar değeri dışarıya verilir ve bir adım üste çıkılır. Bu şekilde ağacın bütün düğümleri gezilir.

Algoritma 1. SıraÖnce(A:ağaç)

```
1.  eğer x≠NIL ise
2.      Yaz(değer(A))
3.      SıraÖnce(sol[A])
4.      SıraÖnce(sağ[A])
```

Algoritma 2. Sıralı(A:ağaç)

```
1-  eğer x≠NIL ise
2-      Sıralı(sol[A])
3-      Yaz(değer(A))
4-      Sıralı(sağ[A])
```

Algoritma 3. SıraSonra(A:ağaç)

```
1-  eğer x≠NIL ise
2-    SıraSonra(sol[A])
3-    SıraSonra(sağ[A])
4-    Yaz(değer(A))
```

Şekil 18.2' de görülen ikili ağaç için verilen üç gezinme algoritmasının sonucu aşağıdaki gibi olur.

SıraÖnce: 6, 4, 1, 5, 9, 7, 8

Sıralı: 1, 4, 5, 6, 7, 8, 9

SıraSonra: 1, 5, 4, 8, 7, 9, 6

Dikkat edilecek olursa, ikili arama ağacına yerleştirilmiş verilerin sıralı verilmesi hiç de zor değildir. Çünkü Sıralı gezinme işleminin sonucu verilerin sıralanmış şekli olur.

Verilen üç algoritmanın çalışma zamanı birbirine çok yakın olacağından dolayı sadece bir tanesinin analizinin yapılması yeterlidir.

Tam bir ikili ağaç olduğu kabul edilsin ve ağacın yüksekliği k olsun. Bu durumda ağaçta toplam düğüm sayısı

$2^0+2^1+2^2+...+2^k=2^{k+1}-1$

olur. n düğüm sayısı olduğundan $n=2^{k+1}-1$ olur. Yapraklar dahil her düğüm üzerinden 2 sefer geçileceğinden dolayı zaman bağıntısı ağaçta bulunan düğüm sayısının lineer bağıntısı olur. Zaman bağıntısı T(n) olmak üzere

$T(n)=T(n-1)+\Theta(1)$

olur. Bu tekrarlı bağıntı asimptotik olarak çözüldüğünde

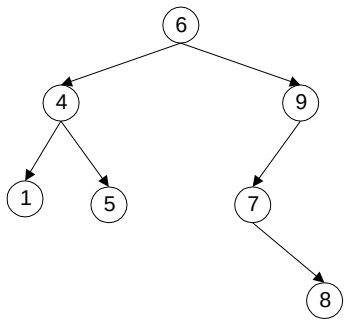
$T(n)= \Theta(n)$

elde edilir.

İkili arama ağaçları ile ilgili hatırda tutulması gereken bir özellik var ve bu özellik:

A bir ikili arama ağacı olmak üzere; x ve y A ağacında düğüm olmak üzere eğer anahtar[x]≤anahtar[y] ise, x düğümü A ağacında y düğümünün solunda yer alır ve y düğümü de x düğümünün sağında yer alır.

İkili ağaçlar üzerinde en çok yapılan işlemlerden biri kayıt arama işlemidir. Bunun yanında bir ikili arama ağacında maksimum, minimum, bir sonraki veya bir önceki eleman aramaları da yapılmaktadır. Burada bu işlemlerin nasıl yapıldığı üzerinde durulacak ve bu yapılan işlemlerin analizi yapılacaktır.



Şekil 2. *İkili ağaç.*

Bir ikili arama ağacında arama yapılması için bir kaydın anahtar değeri verilir ve bu anahtar değerinin ağaçta olup olmadığı kontrol

edilir. Verilen anahtarın değerinden yola çıkarak ikili arama ağacın özelliğinden faydalanarak ağaç üzerinden bir yol izlenir. Bu yol üzerinde bu anahtar ile aynı değere sahip olan düğüm varsa, kayıt ağaçta var demektir ve bu düğümün bütün bilgisi dışarıya verilir. Eğer NIL değerine ulaşılırsa, bunun anlamı kayıt ağaçta yok demektir. İkili arama ağacı üzerinde arama yapan algoritması Algoritma 18.4' te görülmektedir.

Şekil 18.2' de verilen ağaç için bu algoritmanın çalışmasına kısaca göz atılabilir. Ağaçta 8 değeri aranıyor olsun. Algoritma ağacın kök düğümünden başlar ve 8 ile 6 değerini karşılaştırır. 8 değeri 6 değerinden büyük olduğundan ve sağ bağlantı NIL olmadığından sağa hareket edilir ve bir sonraki adımda 9 ile 8 karşılaştırılır. 8 değeri 9 değerinden küçük olduğundan ve sol bağlantı NIL olmadığından dolayı sola hareket edilir. Bundan sonraki adımda 8 değeri ile 7 değeri karşılaştırılır. 8 değeri 7 değerinden büyük ve sağ bağlantı NIL olmadığından dolayı sağa hareket edilir. Son adım olarak sağdaki düğümde 8 değeri olduğundan aranan elemanın değeri de 8 olduğundan karşılaştırmanın sonucu başarılı olacağından bu düğüm aranan kayıt olarak dışarıya verilir. Ağaç üzerinde izlenen yol

$6 \rightarrow 9 \rightarrow 7 \rightarrow 8$

şeklinde olur ve bu bir başarılı aramadır.

Aynı ağaç üzerinde 10 değeri aranıyor olsun. Arama işlemi her zaman olduğu gibi kök düğümünden başlar ve 10 ile 6 değeri karşılaştırılır. Karşılaştırmanın sonucunda 10 değerinin 6 değerinden büyük olduğu ve sağ bağlantının NIL olmamasından dolayı sağa hareket edilir. Bir sonraki düğümün anahtar değeri 9 ve 10 ile 9 değeri karşılaştırılır. 10 değeri 9 değerinden büyük olduğundan dolayı sağa hareket edilir ve NIL değere düşülür. Bunun anlamı aranan 10 değeri bu ağaçta yoktur. Bu da başarısız bir arama olur ve izlenen yol

$6 \rightarrow 9 \rightarrow \text{NIL}$

şeklinde olur.

İkili arama ağacında arama yapmanın temel mantığı ikili arama ağaçlarının özelliğinde yatmaktadır. Aranan elemanın anahtar değeri kök düğümünde bulunan kaydın anahtar değeri ile karşılaştırılır. Eğer aranan kaydın anahtar değeri kök düğümünün anahtar değerinden küçük çıkarsa, bunun anlamı aranan kayıt sağ alt ağaçta olması mümkün değildir ve bundan dolayı sol alt ağaç içerisinde aramaya devam edilmelidir. Aranan kaydın anahtar değeri kök düğümünün anahtar değerinden büyük çıkarsa, bunun anlamı aranan kaydın sol alt ağaçta olması mümkün değildir. Bundan dolayı arama işlemine sağ alt ağaçta devam edilir.

Algoritma 4. İkiliArama(A,x)

▷ A: ikili arama ağacı ve x bu ağaçta aranan kaydın anahtar değeri.

```
1-  eğer A=NIL veya x=anahtar[A] ise
2-      O an üzerinde bulunulan düğümü
          dışarıya ver.
3-  eğer x<anahtar[A] ise
4-      İkiliArama(sol[A],x)
5-  değilse
6-      İkiliArama(sağ[A],x)
```

İkili arama algoritmasının analizinin yapılması için verilen ağacın yüksekliği h olsun. Arama yapılacağı zaman şu ana kadar görüldüğü gibi ağaç içerisinde sadece bir yol izlenmektedir. Başarılı bir arama için yapılacak karşılaştırma sayısı 1 ile h arasında değişir. Aranan kayıt kök düğümünde ise, karşılaştırma sayısı 1 olur. Aranan kayıt kök düğümünün çocuklarından birinde ise, karşılaştırma sayısı 2 olur ve bu şekilde devam edilerek arana kayıt en derindeki yapraklardan birinde ise karşılaştırma sayısı h olur. Arama yapılırken karşılaştırma sayısı farklı h tane arama yapılır. Ortalama karşılaştırma sayısı

$$\frac{1+2+\cdots+h}{h} = \frac{h+1}{2}$$

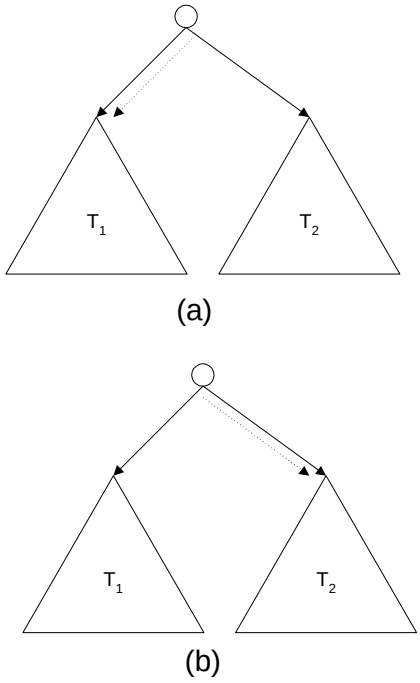
olur. Buradan T(n) zamanı T(n)=O(h)=O(lgn) olur. Başarısız bir arama için her seferinde NIL değerine ulaşılması gerekir. Verilen ağaç tam ağaç değilse, yapılacak karşılaştırma sayıları 2 ile h+1 arasında değişir. Bu durumda ortalama karşılaştırma sayısı

$$\frac{2+3+\cdots+(h+1)}{h} = \frac{(h+1)(h+2)}{h} - 1$$

olur. Buradan T(n) zaman bağıntısı

T(n)=O(h)=O(lgn)

olur.



Şekil 3. *(a) Aranan kaydın anahtar değeri kök düğümünün anahtar değerinden küçük olduğundan sol alt ağaç (T_1) içerisinde arama işlemine devam edilir; (b) Aranan kaydın anahtar değeri kök düğümünün anahtar değerinden büyük olduğundan sağ alt ağaç (T_2) içerisinde arama işlemine devam edilir.*

Algoritma 18.4' te ikili arama ağaçları üzerinde özyinelemeli arama algoritması verilmişti. Bu algoritmanın iteratif şeklide Algoritma 18.5' te görülmektedir.

Algoritma 5. İkiliArama(A,x)

▷ A: ikili arama ağacı ve x bu ağaçta aranan kaydın anahtar değeri.

```
1-  A≠NIL ve x≠anahtar[A] oldukça devam et
2-      eğer x<anahtar[A] ise
3-          A←sol[A]
4-      değilse
5-          A←sağ[A]
6-  A düğümünü dışarıya ver.
```

İkili arama ağaçları üzerinde yapılan özyinelemeli arama ile iteratif arama arasında sonuç bakımından hiçbir fark yoktur. İşleyiş bakımından fark vardır.

İkili ağaçlar üzerinde yapılan diğer arama çeşitleri ise, daha önce belirtildiği gibi minimum, maksimum, bir önceki kayıt ve bir sonraki kayıt aramalarıdır.

Bir ikili ağaçta anahtar değeri diğer kayıtların hepsinden küçük olan kayda minimum kayıt denir. Bu kaydın bulunması için ağaç üzerinde sürekli sola gidilir ve sol bağlantısı NIL olan düğüm bulununcaya kadar devam edilir. Bu düğüm bulunduğunda, bu düğüm içerisindeki kayıt minimum olarak dışarıya verilir. Algoritma 18.5. verilen ikili arama ağacında minimum kaydı içeren düğüme bir işaretçi döndürür.

İkili arama ağacının özelliğinden dolayı minimum kayıt her zaman en soldadır. Bundan dolayı verilen Minimum(A,p) algoritmasının çalışması doğru sonuç verir. Bu algoritmanın analizi yapılacak olursa, arama algoritmasının başarılı araması ile aynı sonucu verecektir. Çünkü minimum kaydı aramak için yapılan karşılaştırmalar 1 ile h arasında değişirler (h verilen ağacın yüksekliğidir).

Algoritma 6. Minimum(A,p)

▹ A: ikili arama ağacı ve p bu ağaçta minimum kaydı içeren düğüme bir işaretçidir.

1- sol[A]≠NIL oldukça devam et
2- A←sol[A]
3- p←A

Eğer ağaç içerisindeki bir x düğümünün sol alt ağacı NIL ise, bu düğümün sağında yer alan düğümlerin hepsinin içermiş oldukları kayıtların hepsi x düğümünde bulunan kayıttan büyüktürler. Eğer x düğümünün sol alt ağacı NIL değilse, sol alt ağaç içerisinde minimum olan kayıt aranır ve bu kayıtta sol alt ağacı NIL olan düğüm içerisinde olur.

Benzer şekilde ağaçta maksimum anahtarlı kaydı bulmak için minimum için yapılan işlemlerin simetriği yapılır. En büyük kayıt her zaman en sağdadır. İkili arama ağacın özelliğinden yararlanarak, x gibi bir düğümün sol alt ağacında bulunan bütün kayıtların anahtar değeri x düğümünün anahtar değerinden küçük olacaktır. Bundan dolayı sağ alt ağaçta maksimum kaydın aranması yapılmalıdır. Eğer x düğümünün sağ alt ağacı NIL değilse, sağ bağlantı kullanılarak sağ alt ağaca gidilir ve gidilen düğümün sağ alt ağacı kontrol edilir. Eğer yine sağ alt ağaç NIL değilse, tekrar sağ bağlantı kullanılarak bir sonraki sağ alt ağacın kök düğümüne gidilir. Bu şekilde devam edilerek sağ alt ağacı (sağ çocuğu) NIL olan düğüm bulunduğunda bu düğüm içerisindeki kaydın anahtar değeri maksimumdur. Bu işlemleri yapan algoritma Algoritma 18.7' de görülmektedir.

Maksimum(A,p) algoritmasının analizi yapıldığında, bunun sonucunun da başarılı arama ile aynı olacağı açıktır. Çünkü aranan maksimum kayıt kök düğümde bulunabileceği gibi, en derindeki yaprakta da bulunabilir. Bundan dolayı yapılan karşılaştırma sayısı, yüksekliği h olan bir ağaç için 1 ile h arasında değişir.

Algoritma 7. Maksimum(A,p)

▷ A: ikili arama ağacı ve p bu ağaçta maksimum kaydı içeren düğüme bir işaretçidir.

```
1- sağ[A]≠NIL olduğu sürece devam et
2-      A←sağ[A]
3- p←A
```

Bazı durumlarda verilen bir ağaçta Sıralı yöntemi ile kayıtların yazılması durumunda bir kayıttan sonra gelen kayda (sıralı dizi içerisinde bir sonraki kayıt) ihtiyaç duyulabilir.

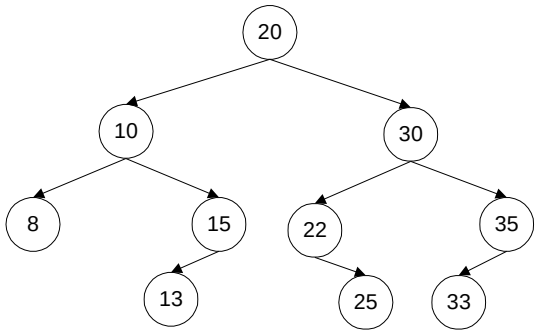
Eğer bir ağaçta bulunan kayıtların hepsinin anahtar değerleri birbirinden farklı ise, x gibi bir düğümde bulunan kaydın bir sonraki kaydı, anahtar değeri anahtar[x]' ten büyük olan ilk kayıttır. İkili arama ağacının yapısı gereği veya özelliği gereği karşılaştırmaya ihtiyaç duymadan bir sonraki kaydın nerede olduğu rahatlıkla bulunabilir. Algoritma 18.8' de gibi bir kaydın varsa bir sonraki kaydını vermektedir; yoksa NIL değerini vermektedir.

Algoritma 8. BirSonra(x,p)

▷ x: ikili arama ağacın bir sonraki kaydı aranan kaydın bulunduğu düğüm; p ise sağ alt ağaç NIL olmadığı durumda bir sonraki kaydın bulunduğu düğüme bir işaretçidir.

```
1-  if sağ[x]≠NIL then
2-       Minimum(sağ[x],p) döndür
3-  y←ata[x]
4-  while y≠NIL ve x=sağ[y] do
5-       x←y
6-       y←ata[y]
7-  y' yi döndür.
```

Örneğin, Şekil 18.4' te verilen ağacın üzerinde bir sonraki kaydı bulmak için x=15 olduğu durumda sağ bağlantı NIL olacağından algoritmanın ilk kısmı çalışmaz. Bundan dolayı ikinci kısmı çalışır ve 15 değerinin bir sonraki kaydı kök düğümde bulunan 20 değeri olur.



Şekil 4. *İkili ağaç.*

Bir sonraki kaydı bulma algoritmasının benzeri olarak bir önceki kaydı bulma işlemi vardır.

Bu algoritmada iki kısımdan oluşmaktadır ve algoritmanın kendisi Algoritma 18.9' da görülmektedir.

Algoritma 9. BirÖnce(x,p)

▷ x: ikili arama ağacın bir önceki kaydı aranan kaydın bulunduğu düğüm; p ise sol alt ağaç NIL olmadığı durumda bir önceki kaydın bulunduğu düğüme bir işaretçidir.

```
1-      eğer sol[x]≠NIL ise
2-          Maksimum(sol[x],p) döndür
3-      y←ata[x]
4-      y≠NIL ve x=sol[y] olduğu sürece devam et
5-          x←y
6-          y←ata[y]
7-      y' yi döndür.
```

Algoritmanın ilk kısmında eğer verilen ağaçta x düğümünün sol alt ağacı NIL değilse, bu sol alt ağaçtaki maksimum kayıt x kaydının bir öncesi olan kayıt olur. Eğer ağaçta x düğümünün sol alt ağacı NIL ise,

bu durumda algoritmanın ikinci kısmı çalışır ve x düğümünün ataları arasında minimum değere sahip olan kayıt x kaydının bir öncesi olur.

Son verilen iki algoritmanın analizi birbirine çok benzemektedir. Çünkü iki algoritma da iki kısımdan oluşmaktadır. İlk kısımlarında daha önce verilen Maksimum ve Minimum algoritmaları kullanıldığından algoritmanın bu kısmının çalışması durumunda mertebeleri Maksimum ve Minimum algoritmaları ile aynı olur.

İkinci kısım çalıştığında ise işaretçilerin ağaç üzerindeki davranışı aşağıdan (yapraklardan köke doğru) yukarıya doğru yapılmaktadır. Bundan dolayı en kötü ihtimalle yapılacak karşılaştırma sayısı 1 ile h-1 arasında olur. Ortalama karşılaştırma sayısı (ağacın yüksekliği h olmak üzere)

$$\frac{1+2+\cdots(h-1)}{h} = \frac{h-1}{2}$$

olur. Buradan bu iki algoritmanın zaman bağıntısının mertebesi

T(n)=O(h)

olur.

Teorem 1

Dinamik küme işlemlerinden arama, minimum, maksimum, bir sonraki ve bir önceki elemanı bulma algoritmalarının mertebeleri, ağacın yüksekliği h olmak üzere O(h) olur ♦

İkili arama ağaçları üzerinde yapılan diğer önemli iki işlem ise, eleman ekleme ve eleman silme işlemleridir. Bu iki işlemin sonucunda ağacın yapısı değişeceğinden dolayı, bu işlemlerin uygulamasının dikkatli yapılması gerekir. Aynı zamanda ağacın yapısı değişirken, ağacın ikili arama ağacı özelliğinin bozulmaması gerekir. Algoritmaları verildiğinde de görüleceği gibi eleman ekleme çok zor bir işlem değildir; fakat bunun yanında eleman silme işlemi oldukça zahmetli bir işlemdir.

T gibi bir ağaca v elemanını eklemeden önce yapısı, T ağacının bir düğümü ile aynı olan bir düğüm oluşturulur ve bu düğüm x olsun.

anahtar[x]=v
sol[x]=NIL
sağ[x]=NIL

şeklinde değer ataması yapılır ve T ağacı içerisinde uygun olan yere eklenir. Eleman ekleme yaprak seviyesinin bir alt seviyesinde yapılır. Yeni bir eleman eklendiği zaman ağacın yüksekliğinin artma ihtimali vardır. Algoritma 18.10' da T ağacına x düğümünü ekleme algoritması görülmektedir.

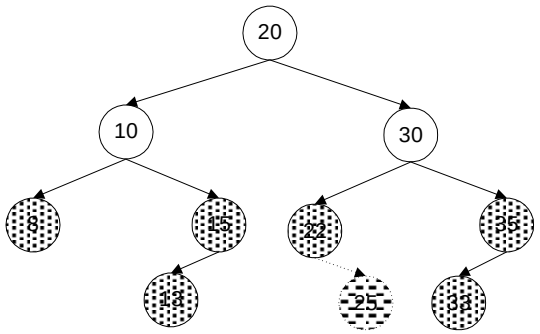
Algoritma 10. Ekleme(T,x)

```
1-      y←NIL
2-      z←kök[T]
3-      x≠NIL olduğu sürece devam et
4-          eğer anahtar[x]<anahtar[x] ise
5-              z←sol[z]
6-          değilse
7-              z←sağ[z]
8-      ata[x]←y
9-      eğer y=NIL ise
10-         kök[T]←x
11-     değil ve eğer anahtar[x]<anahtar[y] ise
12-         sol[y]←x
13-     değilse
14-         sağ[y]←x
```

Daha önce verilen algoritmalarda olduğu gibi eleman ekleme algoritması da kök düğümünden başlar ve yapraklara doğru ilerler.

Şekil 18.5' te görüldüğü gibi 25 değeri bu ağaca eklenecek kayıttır ve yeri de ağaç üzerinde düğümün sınırları kesik çizgi ile gösterilen düğümdür. Diğer içleri dolu olan düğümlerin hepsinin ya bir çocuğu ya da iki çocuğu birden yoktur. Bundan dolayı buralar potansiyel eleman ekleme yerleridirler.

Eleman eklemek için başarısız aramada yapılan karşılaştırma sayısının aynısı bu algoritma içinde geçerlidir. Çünkü ağaca eklenecek olan kayıt için yer bulunduğunda bu yer NIL değeri taşıyan bir yerdir. Bunun anlamı, karşılaştırmalar sonucunda NIL değere düşülmüştür. Bu da başarısız arama ile aynıdır.

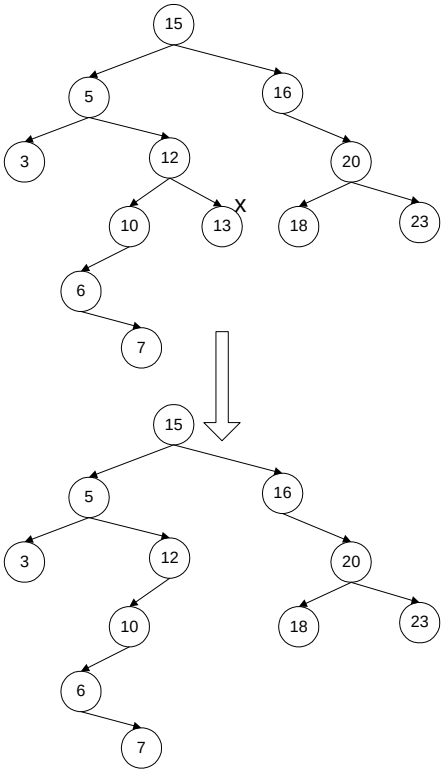


Şekil 5. *T ikili arama ağacına eleman ekleme.*

İkili arama ağacında bir x elemanının silinmesi için; eleman silme algoritması x elemanına bir işaretçi alır. Şekil 18.6' da görüldüğü gibi bir elemanın silinmesi için ikili arama ağacında ilk durum görülmektedir. İlk durumda silinecek düğüm x olmak üzere eğer x düğümünün çocuğu yoksa, sadece o düğüm doğrudan silinir. Ve silinen düğümün atasının x düğümüne işaret eden bağlantısı NIL değerini alır, yani ata[x]=NIL olur.

İkinci durum olarak eğer silinecek olan x düğümünün bir tane çocuğu varsa, bu durumda x düğümünün atasından x düğümünün çocuğuna bir işaretçi oluşturulur ve x düğümü ağaçtan silinir. Şekil 18.7' de silinecek olan düğümün bir tane çocuğu vardır. Görüldüğü gibi x düğümünün atasından x düğümünün çocuğuna bir işaretçi oluşturulmuştur.

Üçüncü durum olarak x düğümünün iki tane çocuğu olduğu durumdur. x düğümünün bir sonraki düğümü bulunur ve bu düğüm y olsun.

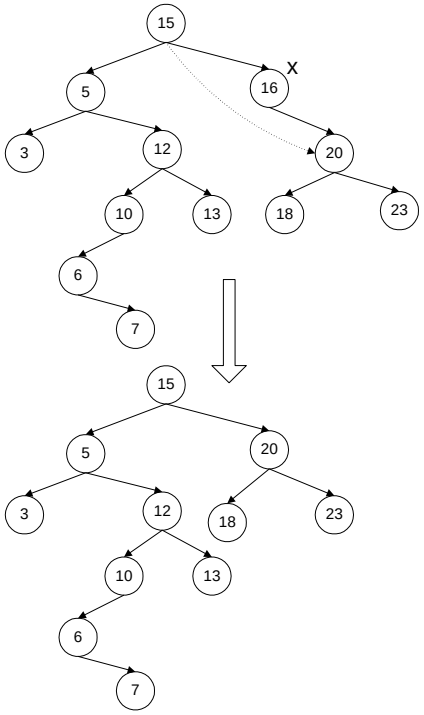


Şekil 6. *x düğümünün ağaçtan silinmesi. Eğer x düğümünün çocuğu yoksa, sadece o düğüm doğrudan silinir.*

y düğümünün içeriği yerine x düğümünün içeriği yazılır ve ağaçtan x düğümü silinir. Bu durum Şekil 18.8' de görülmektedir.

Algoritma 18.11 algoritması x düğümünün bulunabileceği üç durumu da göz önüne alarak ona göre silme işlemini gerçekleştirmektedir.

Son durumda önemli bir nokta vardır; bu nokta x düğümünün bir sonraki düğümü bulunurken, bu düğümün sol çocuğunun olmaması şartının göz önüne alması gerekir. Bundan dolayı, bu nokta önemlidir ve algoritma bu durumu göz önüne almaktadır.



Şekil 7. *x düğümünün bir tane çocuğu vardır ve bu x düğümü silinirse, x düğümünün atasından x düğümünün çocuğuna bir işaretçi oluşturulur.*

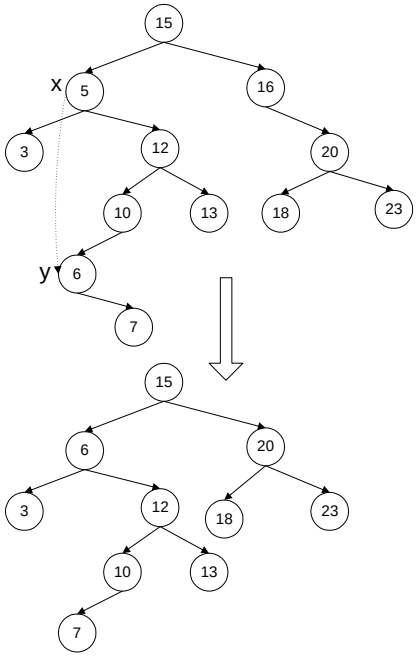
Teorem 2

Yüksekliği h olan bir ikili arama ağacında düğüm ekleme ve düğüm silme algoritmalarının mertebeleri O(h) olur ♦

Düğüm ekleme algoritmasının analizi daha önce yapıldı. Burada ise, düğüm silme algoritmasının analizi yapılacaktır. Bir düğüm silinirken en fazla olacak durum, silinecek olan düğümün iki tane çocuğunun olduğu üçüncü durumdur. Bu durumun analizi yapıldığında diğer iki durum için üst sınır tayin edilmiş olur.

Algoritma 11. Silme(T,x)

```
1-      eğer sol[x]=NIL veya sağ[x]=NIL ise
2-          y←x
3-      değilse
4-          y←BirSonra(x,p)
5-      eğer sol[x]≠NIL ise
6-          z←sol[y]
7-      değilse
8-          z←sağ[y]
9-      eğer z≠NIL ise
10-         ata[z]←ata[y]
11-     eğer ata[y]=NIL ise
12-         kök[T]←z
13-     değil ve eğer y=sol[ata[y]] ise
14-         sol[ata[y]]←z
15-     değilse
16-         sağ[ata[y]]←z
```

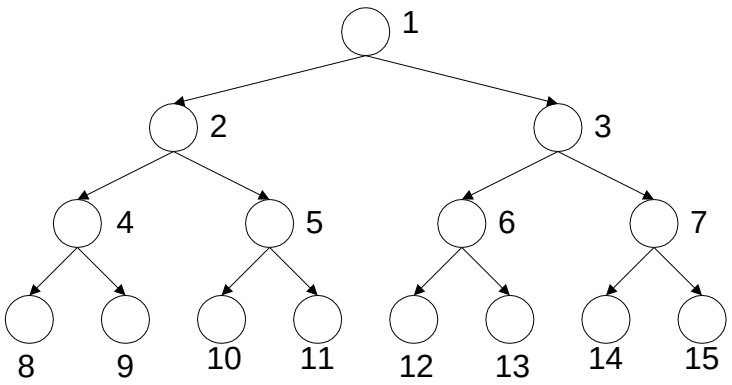


Şekil 8. *y düğümünün içeriği yerine x düğümünün içeriği yazılır ve ağaçtan x düğümü silinir.*

Silinecek düğümün ağaçta bulunması işlemi arama işlemidir ve arama algoritmasının mertebesinin yüksekliği h olan bir ikili arama ağacı için O(h) olduğu bilinmektedir. Bu düğüm bulunduktan sonra yapılacak işlem bir sonraki düğümün bulunması durumudur. Bir sonraki düğümü bulan algoritmanın mertebesi de O(h) olduğundan bundan dolayı silme işleminin mertebesi O(h) olur.

İkili Yığınlar

İkili ağaçlar, bağlı listelerin belli bir düzen içerisinde ve gelen verinin karakteristiğine göre inşa edilen bir veri yapısıdır. Statik verilerle ikili ağacın taklit edilmesi ile oluşturulan veri yapısına ikili yığın denilmektedir ve bu veri yapısının en çok kullanıldığı yerler sıralama işlemi ve öncelikli kuyruk işlemleridir.



Şekil 9. *15 düğümle bir tam ikili ağaç.*

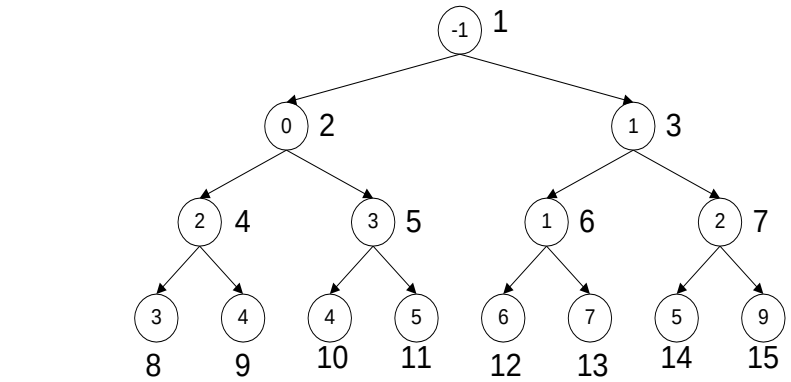
Şekil 18.9' da verilen ikili ağacın düğümleri seviye baz alınarak soldan sağa numaralandırılmıştır. Kök düğümün almış olduğu numara sol çocuğun numarasının yarısı ve sağ çocuğun numarasının 1 eksiğinin yarısıdır.

Bir düğümün numarası k olmak üzere sol çocuğun numarası 2k ve sağ çocuğun numarası ise 2k+1 olur. Eğer bu numaralar dizinin sınırları dışında iseler, bu çocuklar var olmayacaktır.

Bu bilgilerden yola çıkılarak ikili yığın veri yapısının tanımı yapılabilir.

Tanım 3.
İkili yığın, her düğümünde bir anahtar olan bir ikili ağaçtır; öyleki

a) Bütün yapraklar bir seviyede veya en kötü durumda bitişik iki seviyededirler.
b) Her seviye soldan sağa doğru dolmaya başlar. Eğer seviye dolu değilse, o seviyede olan yapraklar sola yanaşık olurlar.
c) Kök düğümde olan anahtar değeri kendi çocuklarının anahtar değerlerinden büyük veya eşittir. Sağ ve sol alt ağaçlarda aynı zamanda ikili yığındırlar.



-1	0	1	2	3	1	2	3	4	4	5	6	7	5	9

Şekil 10. *İkili yığın ve ait olduğu dizi.*

İkili yığın üzerinde işlem yapmak için bazı terimlerin açıklamalarının verilmesi gerekir. A ikili yığın özelliğine göre oluşturulmuş bir dizi olmak üzere uzunluk[A] terimi A dizisi içindeki eleman sayısını ifade eder ve YığınBoyut[A] terimi A dizisi içinde yığın özelliğine göre yerleştirilmiş eleman sayısını ifade eder. Genellikle

YığınBoyut[A]≤Uzunluk[A] eşitsizliği sağlanır. i endeksinin ata düğümü Ata(i), sol çocuk Sol(i) ve sağ çocuk Sağ(i) şeklinde gösterilir.

Ata(i) fonksiyonu $\lfloor i/2 \rfloor$ değerini geri döndürür, Sol(i) fonksiyonu 2i değerini geri döndürür ve Sağ(i) fonksiyonu da 2i+1 değerini geri döndürür. Yığın özelliği olarak bilinen

A[Ata[i]]≥A[i]

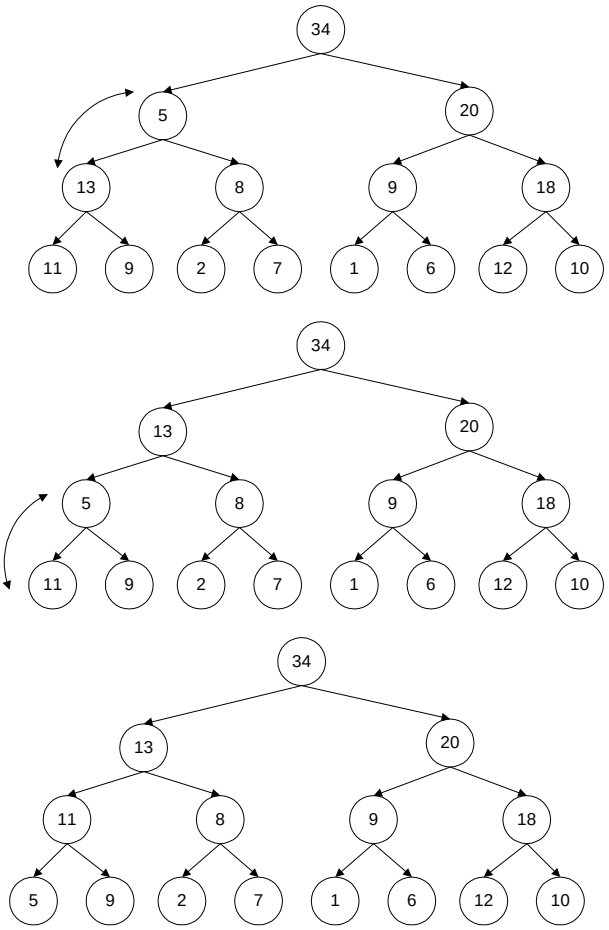
özelliği sağlanmalıdır. Böylece en büyük değer kök düğümde saklanmıştır. Bir düğümün yüksekliği, o düğümden yapraklara doğru aşağı inilirken, üzerinde geçilen ayrıt sayısıdır. Ağacın yüksekliği ise kök düğümden yapraklara olan yoldaki ayrıt sayısıdır. Yığın içindeki eleman sayısı n olmak üzere yığının yüksekliği Θ(lgn) olur ve bu yığın üzerinde yapılacak işlemlerin çoğunun mertebesi O(lgn) olur.

İlk olarak verilen bir dizinin yığın haline getirilmesi işlemi için algoritmanın verilmesidir. Bu işlem için tasarlanan algoritma Algoritma 18.12' de görülmektedir ve bu algoritmanın parametreleri A yığın olmayan bir dizi ve i ise bir düğüm endeksidir. YığınYap() algoritması çağrıldığı zaman Sol(i) ve Sağ(i) düğümlerini kök olarak alan alt ağaçlar yığın özelliğine sahiptir ve A[i] değerinin doğru yere yerleştirilmesi işlemini gerçekleştirir.

Algoritma 12. YığınYap(A,i)

```
l←Sol(i)
r←Sağ(i)
eğer l≤YığınBoyut[A] ve A[l]>A[r] ise
    Büyük←l
 değilse
    Büyük←i
eğer r≤YığınBoyut[A] ve A[r]>A[Büyük] ise
    Büyük←r
eğer Büyük≠i ise
    A[i]↔A[Büyük]
YığınYap(A,Büyük)
```

Şekil 18.11’ de görüldüğü gibi yığın özelliğini bozan anahtar değeri çocukların anahtar değerleri ile karşılaştırılır. Hangisi ile uyuşmazlık içerisinde ise, onunla yer değiştirir. Yığın özelliğini bozan anahtar değeri yaprağa kadar inilinceye kadar veya yığın özelliği sağlanıncaya kadar işleme devam edilir. Yığın özelliği sağlanınca YığınYap algoritması bitime gider. Dizi içerisinde n tane anahtar değeri varsa, bu durumda yığın ağacının yüksekliği $\lceil \lg n \rceil$ olur.



Şekil 11. *2 nolu düğümde yer alan anahtar değeri 5 yığın özelliğini bozmaktadır ve YığınYap işleminin çalışma şekli.*

YığınYap algoritması özyinelemeli bir algoritma olduğundan zaman bağıntısı tekrarlı bağıntı olur ve

$T(n) \leq T(2n/3)+\Theta(1)$

olur. Bu tekrarlı bağıntı Master yöntemine göre çözüldüğü zaman $n^{\log_{\frac{3}{2}} 1} = n^0 = 1$ olduğundan $T(n)=\Theta(\lg n)$ olur. Klasik olarak açıklanacak olursa, ağacın yüksekliği $\Theta(\lg n)$ şeklindedir ve yığın yapma işleminde ağacın yüksekliğine yakın sayıda karşılaştırma yapılır. Bu durumda da algoritmanın mertebesi $\Theta(\lg n)$ olur.

Bir diziyi yığın haline getirmek için YığınYap algoritmasının dizideki düğüm sayısının yaklaşık yarısı kadar çalıştırılması gerekiyor ve bu işlemi yapan algoritma Algoritma 18.12' de görülmektedir.

Algoritma 13. YığınİnşaEt(A)

```
YığınBoyut[A]←Uzunluk[A]
i←⌊Uzunluk[A]/2⌋ ... 1 kadar
   YığınYap(A,i)
```

Algoritma 18.12' de dizinin yarısı kadar işlem yapılmasının sebebi A[$\lfloor n/2 \rfloor$+1 ... n] elemanları yaprakları teşkil ederler. Geriye kalan düğümlerin hepsi için birer kez YığınYap algoritması çağrılır.

YığınİnşaEt algoritması her seferinde YığınYap algoritmasını çağırır ve çağrılan algoritmanın mertebesi $\Theta(\lg n)$ olduğundan YığınİnşaEt algoritmasının mertebesi $O(n \lg n)$ olur.

Daha kesin bir sınır elde edilebilir. Bir yığın içinde h yüksekliğinde en fazla $\lceil n/2^{h+1} \rceil$ tane düğüm vardır. Yüksekliği h olan bir düğüm için YığınYap algoritması çağrıldığında yapılan işlemin mertebesi O(h)

olur. Böylece toplam maliyet YığınİnşaEt algoritması için hesaplanabilir ve

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right)$$

olur. Toplam sembolu içerisindeki değer üslü ifadeye dönüştürüldüğünde 1' den küçük bir değer olacağından bu toplam bir geometrik seridir ve sabit bir sayıya gider. Bundan dolayı YığınİnşaEt algoritmasının mertebesi O(n) olur.

İkinci bir yolla bu algoritmanın mertebesi hesaplanacak olursa, en kötü durum analizi için tekrarlı bağıntı

$W(n)=W(n-r-1)+W(r)+2\lg n$

olur. Yığın dengeli bir ağaç olduğundan r en fazla n/3 değerini alır. Tam bağlı ikili ağaç için $N=2^h-1$ olur ve n değeri N/2 ile N arasında olur. Bu durumda W(N) bağıntısı W(n) için bir üst sınır olur. $N=2^h-1$ için tekrarlı bağıntı

$W(N)=2W((N-1)/2)+2\lg N$

olur. Bu durumda Master yöntemi uygulanabilir. $f(N)=2\lg N$ olur ve $N^{\log_2 2} = N$ olur ve $\varepsilon=0.1$ olarak alınırsa $2\lg N \in O(N^{0.9})$ olur. Algoritmanın mertebesi $W(N)=\Theta(N)$ olur.

YığınİnşaEt algoritması kullanılarak A dizisi sıralanabilir. A dizisinin sıralama işlemini yapan algoritma Algoritma 18.14' te görüldüğü gibi olur.

Bu algoritmanın analizi yapılırken, iki bölümde ele almak gerekiyor. Birinci bölümde YığınİnşaEt algoritmasının mertebesi ele alınacaktır ve bu mertebenin lineer olduğu daha önce gösterildi. Algoritmanın geriye kalan kısmında ise n-1 defa YığınYap algoritması çağrılmaktadır ve bu çağrılan algoritmanın mertebesi O(lgn) olduğu da daha önce gösterildi. Bu durumda sıralama işleminin mertebesi

$T(n)=O(nlgn)+O(n)$
$=O(nlgn)$

olur.

Algoritma 14. YığınSırala(A)

```
YığınİnşaEt(A)
İ←Uzunluk[A] ... 2 kadar
   A[1]↔A[i]
   YığınBoyut[A] ←YığınBoyut[A]-1
    YığınYap(A,1)
```

Teorem 3
YığınSırala algoritmasında en kötü durumda yapılan karşılaştırma sayısı 2nlgn+On) olur. Böylece YığınSırala algoritmasının mertebesi Θ(nlgn) olur.

İspat
Yığın inşa etme kısmında en fazla O(n) tane karşılaştırma yapılır ve silme kısmında en fazla 2nlgn tane karşılaştırma yapılır ♦