BÖLÜM 20. B-AĞAÇLARI

Veri yapısı olarak B-ağaçları, B*-ağaçları, B+-ağaçları adları altında birbirlerinden az farklılık gösteren yapılar kullanılmaktadır. Bunlar aynı zamanda dosya yapısı olarak da kullanılabilmektedir. İkili ağaçlar da dosya yapısı olarak kullanılabilir, fakat bu ağaçlarda sadece bir kayıt araması yapılabilir. B-ağaçlar da ise anahtar değeri belli bir aralığa düşen kayıtlar aranabilir. Bu bölümde veri yapısı olarak B-ağaçları üzerinde daha çok durulacaktır; dosya yapısı olarak B-ağaçlarına daha az yer verilecektir.

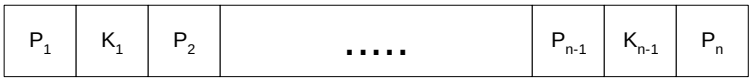
Birincil hafızanın hızı ikincil hafızaya göre çok yüksek olduğundan bir kelimeyi birincil hafızada okumak nanosaniyeler seviyesindeyken ikincil hafızada bir kelimeyi okumak milisaniye seviyelerine kadar çıkabilmektedir. Diğer bir önemli nokta da diskte bir kelime yerine bir blok veya bir sayfa okunduğundan mümkünse okunan bütün bilginin istenen bilgi olmasını sağlamak veya en azında büyük bir kısmının istenen bilgi olmasını sağlamaktır.

Disk işlemlerinin zamanı çok büyük olduğundan mümkün mertebe disk işlemlerinin en aza indirilmesi gerekir. Bundan dolayı bir blok okuma veya bir sayfa okuma işlemi yapıldığında gerekli bilgi bu sayfa veya blok içerisinde ise bir daha diske gitme ihtiyacı duyulmaz. Bu şekilde disk işlemleri azaltılmış olur.

İkili ağaçlar birincil hafızadaki veriler için kullanılan bir veri yapısıdır ve sadece tek kayıt aramaya yönelik tasarlanmış bir veri yapısıdır. Belli bir aralığa düşen veri grubunu aramada etkili bir veri yapısı değildir. Bundan dolayı ikili ağaçlar çok-yollu ağaçlar haline getirilir. Çok yollu ağaçlarda bütün yapraklar aynı seviyede ve bütün düğümler için düğüm derecesi belli bir aralıkta ise, bu elde edilen ağaca B-ağacı (B+-ağacı, B*-ağacı) ismi verilmektedir.

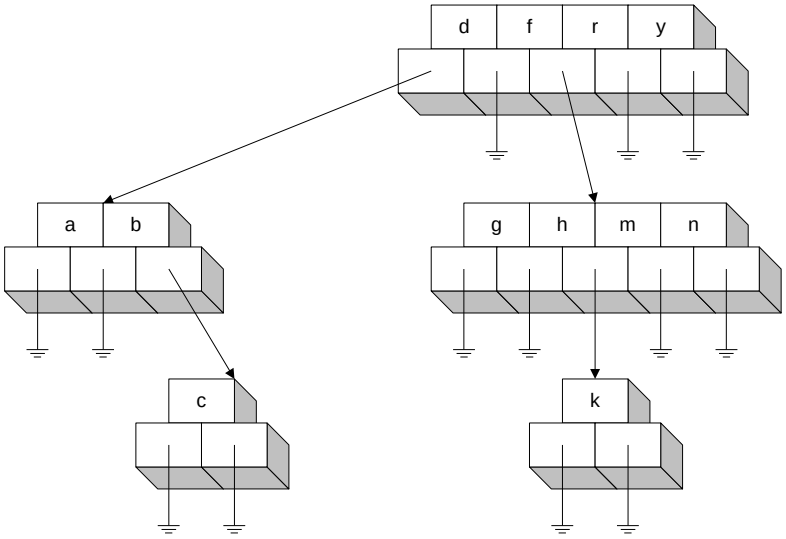
B-ağaçlarında her düğümde birden fazla anahtar değeri saklanır ve her anahtar değeri için bir işaretçi bir sonraki seviye için tutulur. Ağacın mertebesi tanımlanır ve her düğüm en fazla ağacın mertebesi kadar

çocuğa sahip olabilir. Şekil 20.1' de bir B-ağacının düğümü görülmektedir.



Şekil 20.1. *Bir çok-yollu arama ağacının düğüm şekli.*

P_i ($1 \le i \le n$) sembolleri işaretçileri ifade ederken, K_j ($1 \le j \le n-1$) sembolleri ise anahtarları ifade etmektedir. Eğer bir düğümde $k \le m$ tane anahtar varsa ve ağacın mertebesi m ise, bu düğümde en fazla 2m tane anahtar olabilir.



Şekil 20.2. *5-yollu arama ağacı.*

Şekil 20.2' de verilen 5-yollu arama ağacında iki tane düğümü tam doludur ve diğer düğümlerin bir kısmı boş iken bir kısmı da kısmen doludur. Bu bölümün ilk kısımlarında değinildiği gibi disk işlemleri çok zaman aldığından disk erişimlerinin minimum olması sistemin

performansını arttırır. Bundan dolayı çok-yollu ağacın yüksekliğinin mümkün olduğunca küçük olması gerekir. Bu şekilde disk erişimleri minimize edilmiş olur. Minimum yüksekliğe erişebilmek için yaprak seviyesi hariç diğer seviyelerde boş düğümün olmaması gerekir ve düğüm dolulukları ile ilgili olarak da bir sınırlamanın olması gerekir. Bu şekilde bütün yapraklar aynı seviyede olurlar. Yaprak ve kök düğümleri hariç dahili (ara) düğümlerin hepsi en az sahip olacakları çocuk sayısının en fazla sahip oldukları çocuk sayısının yarısı kadar olduğu açıktır. Bütün bu bilgilerden sonra elde edilecek çok-yollu arama ağacına B-ağacı deniliyor ve tanımı aşağıdaki gibi yapılır.

Tanım 20.1

Bir B-ağacı köklü bir ağaç olup aşağıdaki özelliklere sahiptir.

1- Her x düğümü şu özelliklere sahiptir.
   a) Kök düğümü en az bir tane kayda sahip olabilir.
   b) n[x] tane anahtar azalmayacak sırada yerleşmiştir: $anahtar_1[x] \le anahtar_2[x] \le ... \le anahtar_{n[x]}[x]$ olur.
   c) yaprak[x] bir boolean değişken olup eğer x bir yaprak düğüm ise değeri 1 (true) ve yaprak düğüm değilse değeri 0 (false) olan bir değişkendir.
2- Eğer x bir dahili düğüm ise, n[x]+1 tane işaretçiye sahiptir ($c_1[x]$, $c_2[x]$, ..., $c_{n[x]+1}[x]$) ve her işaretçi bir çocuğu işaret etmektedir. Yaprak düğümlerin çocukları yoktur ve işaretçileri tanımsızdır.
3- B-ağacının mertebesi v olarak verildiyse, bütün düğümler en fazla 2v tane kayıt içerebilirler ve kök düğümü hariç diğer düğümler en az v tane kayıt içerirler.
4- Bütün yapraklar aynı seviyede yer alırlar ve her yaprağın sahip olduğu derinlik ağacın yüksekliğini (h) verir.
5- Bir düğümün sahip olabileceği anahtar sayısının alt ve üst sınırı vardır. Alt sınır B-ağacı için **minimum derece** (t olsun) olarak ifade edilir. Minimum derece bir düğümün sahip olabileceği minimum çocuk sayısıdır. Derece ise bir düğümün o anki çocuk sayısıdır. Aynı zamanda minimum derece (kök düğüm hariç) B-ağacının mertebesinin bir fazlasıdır.
   a) Kök düğümü hariç diğer düğümlerin hepsi en az t-1 tane düğüme sahip olmalıdırlar. Kök düğümü hariç dahili düğümler

en az t tane çocuğa sahip olurlar. Eğer ağaç boş değilse, kök düğümü en az bir tane anahtara sahiptir.

b) Her düğüm en fazla 2t-1 tane anahtara sahip olurlar. Böylece dahili düğümler en fazla 2t tane çocuğa sahip olurlar. Eğer bir düğümde 2t-1 tane anahtar varsa, o düğüm **dolu** olarak tabir edilir.

B-ağaçları için daha sade olan bir tanım aşağıdaki gibi yapılabilir.

Tanım 20.2

Mertebesi m olan bir B-ağacı m-yollu arama ağacı olup

a) Bütün yaprakları aynı seviyededir.

b) Kök düğümü hariç diğer bütün dahili düğümler en fazla m tane boş olmayan çocuğa sahiptir ve en azda $\lceil m/2 \rceil$ tane boş olmayan çocuğa sahiptirler.

c) Dahili düğümlerdeki anahtar sayısı o düğümün sahip olduğu çocuk sayısının bir eksiğidir. Bu anahtarlar çocuklarda bulunan anahtarları belli aralıklara göre gruplara ayırırlar.

d) Kök düğümü en fazla m tane çocuğa sahip olabilir. Eğer kök düğümü aynı zamanda yaprak değilse en az 2 tane çocuğa sahiptir.

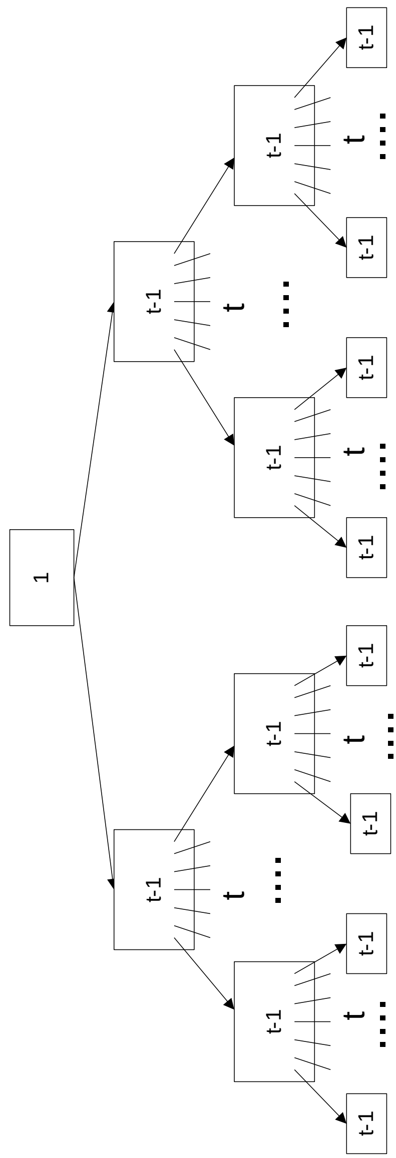
Eğer B-ağacı yapısı bir dosya yapısı olarak kullanılıyorsa, disk erişim sayısı ağacın yüksekliği ile orantılıdır. En kötü durumdaki ağacın yüksekliği bundan dolayı önemlidir.

Teorem 20.1

Eğer $n \geq 1$ ise, yüksekliği h ve minimum derecesi $t \geq 2$ olan n-anahtarlı B-ağacı T için

$$h \leq \log_t \frac{n+1}{2}$$

olur.



Şekil 20.3. *Yüksekliği 3 olan ve minimum sayıda anahtar içeren bir B-ağacı.*

İspat

Eğer bir B-ağacında kök düğümü 1 tane anahtar ve geriye kalan bütün düğümler t-1 tane anahtar içeriyorlarsa, bu ağaç bu yükseklik ve bu kadar düğüm sayısı ile minimum anahtara sahiptir. Bu durumda 0 derinliğinde 1 tane düğüm (kök düğümü), 1 derinliğinde 2 tane düğüm

vardır; 2 derinliğinde 2t tane düğüm vardır ve son olarak 3 derinliğinde $2t^2$ tane düğüm vardır. Eğer ağacın yüksekliği h ise, bu ağacın son seviyesindeki düğüm sayısı $2t^{h-1}$ olur. Bu ağacın içermiş olduğu anahtar sayısı niçin

$$
\begin{aligned}
n &\geq 1 + 2(t-1)\sum_{k=1}^{h} t^{k-1} \\
&= 1 + 2(t-1)\frac{t^h - 1}{t-1} \\
&= 2t^h - 1
\end{aligned}
$$

yazılabilir ♦

B-ağaçlarının inşa edilmeleri sırasında bütün yaprakların aynı seviyede ve dahili düğümlerin sahip oldukları anahtarlarla ilgili alt ve üst sınır şartları yerine getirilir. B-ağacının inşa edilme işleminden önce bu işlemde kullanılan B-ağaçlarında kayıt arama işleminin anlatılması gerekir.

B-ağacında arama yapmak, ikili arama ağacında arama yapmaya çok benzer. İkili arama ağacında arama yaparken, üç durum söz konusudur; ya aranan kayıt o an üzerinde bulunan düğümdeki kayıt ile aynıdır veyahut aranan kaydın anahtar değeri o an üzerinde bulunulan kaydın anahtar değerinden küçüktür veya büyüktür. Bu üç duruma göre işlemler yapılır, fakat B-ağacında ise, durum sayısı ağacın mertebesi ile orantılı olarak değişir. Diğer bir deyişle ikili arama ağacı 2-yollu bir arama ağacıdır, fakat B-ağacı ise, bir (n[x]+1)-yollu arama ağacıdır.

İkili arama ağacı üzerinde yapılan arama işleminden esinlenerek B-ağacı üzerinde arama işleminin algoritması gerçekleştirilebilir. Gerçekleştirilecek algoritmanın iki tane parametresi olacaktır ve bu parametrelerden biri B-alt ağacının kök düğümüne işaret eden bir işaretçi ve diğeri de aranacak kaydın anahtar değeri k' dir. Eğer yapılan arama işleminin sonucu başarılı ise, (y,i) şeklinde bir ikili geriye döndürülür ve y kaydın bulunduğu düğüm ve i ise kaydın

düğüm içerisindeki endeksidir; $anahtar_i[y]=k$ olur. Eğer kayıt ağaçta yoksa, geriye NIL değeri döndürülür.

Algoritma 20.1. B-Ağacı-Arama(x,k)

▹Bu algoritma, bir B-ağacında verilen kaydın olup olmadığını kontrol eder. Eğer kayıt varsa, geriye kaydın bulunduğu düğüme bir işaretçi ve kaydın düğüm içerisindeki endeksi döndürülür. Eğer kayıt B-ağacında yoksa, geriye NIL değeri döndürülür.

```
                i←1
                i≤n[x] ve k>anahtar_i[x] olduğu sürece devam et
  i←i+1
eğer i≤n[x] ve k=anahtar_i[x] ise
    (x,i) ikilisini geriye döndür
eğer yaprak[x] ise
   geriye NIL döndür
değilse
   B-Ağacı-Arama(c_i[x],k)
```

Bu algoritma üç bölümden oluşmaktadır. Birinci bölümü bir düğüm içerisinde ardışıl arama yapmaktadır. Ondan sonra eğer istenilen kayda ulaşılmışsa, sonuç geriye döndürülmektedir ve eğer yaprak düğüme ulaşılmış ve hala kayda rastlanılmamışsa, başarısız arama olarak nitelendirilmektedir. Son bölümde ise, uygun alt ağaca dallandıktan sonra tekrar arama işlemine devam edilmektedir.

Bu algoritmada hem ardışıl arama varken hem de ikili arama ağacına benzer ağacın kök düğümünden yapraklara doğru bir arama vardır. Yapraklara doğru olan arama zaman bağıntısının mertebesi, h ağacın yüksekliği olmak üzere $\Theta(h)=\Theta(\log_t n)$ olur. t bir düğümün (kök düğümü hariç) sahip olabileceği minimum çocuk sayısı ve n ise ağaçta bulunan toplam kayıt sayısıdır. Bir düğüm içerisinde yapılan ardışıl

aramanın mertebesi ise O(t) olduğundan toplam zaman bağıntısının mertebesi $O(th)=O(t\log_t n)$ olur.

B-ağacını inşa etmek için ilk olarak boş bir kök üretilir. Ondan sonra bu kök düğümüne kapasitesi el verdiği sürece kayıt eklemesi yapılır. Kök düğüm dolduktan sonra parçalanma meydana gelir ve bunun sonucunda iki seviyeli ağaç meydana gelir. Ağacın parçalanmasını, B-ağacının sınırlamalarına göre gerçekleştiren ekleme algoritmasıdır. Bir B-ağacını oluşturmak üç basamaktan oluşmaktadır. Birinci adım boş bir kök düğümünün oluşturulmasıdır. İkinci adım ise, bu düğüme veya diğer düğümlerde olsa ağaçta düğüm sayısı değişmeden kaydın eklenmesidir. Son olarak da eğer kaydın denk geldiği yaprakta boş yer yoksa, ağaçta düğüm sayısı artar. Yani ağaçta parçalanma meydana gelir. Bu algoritmalar Algoritma 20.2, Algoritma 20.3 ve Algoritma 20.4' te sırasıyla görülmektedir.

Algoritma 20.2. Boş-Kök-Düğüm(T)

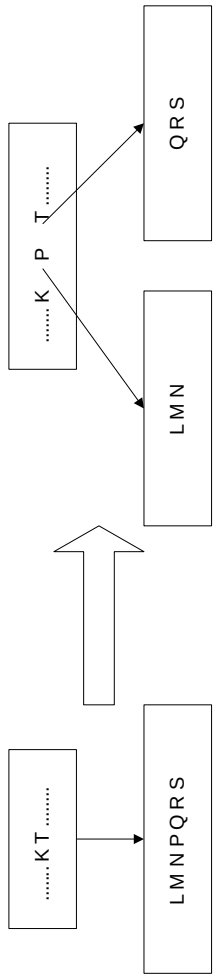
▷ T bir B-ağacıdır. Fakat hafızada henüz böyle bir ağaç yoktur. Bu algoritma, bu ağaç için boş bir kök düğüm oluşturacaktır.

```
x←Düğüm_Al()
yaprak[x]←1
n[x]←0
kök[T]←x
```

Bu algoritmanın mertebesi açıkça görülmektedir ki O(1) olur. Çünkü yapılan işlem veri boyutundan bağımsızdır. Veri boyutu ne olursa olsun yapılan işlem sabittir. Düğüm_Al() fonksiyonu hafızadan bir düğümlük yer ayırır ve bu yerin adresi x değişkenine aktarılır.

Oluşturulan bir B-ağacına bir kayıt eklemek, ikili ağaca kayıt eklemekten çok komplike bir işlemdir. Kayıt ekleme sırasında kaydın denk geldiği düğüm dolu ise (yani 2t-1 tane kayda sahipse), bu düğüm, medyan anahtar $anahtar_t[x]$ civarında ikiye bölünür. Elde

edilen sol düğüm t-1 tane anahtara sahip olur, sağ düğüm ise t tane anahtara sahip olur ve medyan anahtar bir seviye yukarı çıkar. Eğer yukarısı dolu değilse, bu anahtar bu düğüme yerleştirilir. Eğer dolu ise, bölme işlemi bu düğüme de uygulanır ve bu düğümün medyanı bir üst seviyeye gönderilir.



Şekil 20.4. *Bir düğümün bölünmesi ve o düğümün medyan anahtarı olan P anahtarının bir seviye yukarı kaydırılması.*

Algoritma 20.3. Düğüm_Bölme(x,i,y)

▷ kendisine parametre olarak gelen düğümün bölünmesi işlemini yapar.

```
z←Düğüm_Al()
yaprak[z]←yaprak[y]
n[z]←t-1
j←1...t-1 kadar
    anahtar_j[z]←anahtar_{j+t}[y]
eğer yaprak[y] değilse
    j←1...t kadar
        c_j[z]←c_{j+t}[y]
n[y] ←t-1
j←n[x]+1 ... i+1 kadar
    c_{j+t}[x]←c_j[x]
c_{i+1}[x] ←z
j←n[x] ... i kadar
    anahtar_{j+1}[x] ←anahtar_j[x]
anahtar_i[x] ←anahtar_t[y]
n[x] ←n[x]+1
```

Düğüm bölme algoritmasının mertebesini elde etmek için algoritma analiz edildiğinde kullanılan döngülerin hepsi düğüm içerisindeki anahtar sayısı bazında yapılmaktadır. Bundan dolayı bu algoritmanın mertebesi $\Theta(t)$ olur.

Eğer r kök düğümü dolu ise, s gibi boş bir düğüm oluşturulduktan sonra kök düğümü ikiye bölünüyor ve elde edilen iki düğüm s düğümünün çocukları olarak atanıyorlar. T ağacının yeni kök düğümü s düğümüdür. Düğüm_Ekle_DoluDeğil algoritması x dolu olmayan düğümüne k kaydını eklemektedir.

Algoritma 20.4. Düğüm_Ekleme(T,k)

▷ Bu algoritma T ağacına k anahtarını ekleyen bir algoritmadır.

```
r←kök[T]
eğer n[r]=2t-1 ise
   s←Düğüm_Al()
   kök[T] ←s
   yaprak[s] ←0
   n[s] ←0
   c1[s] ←r
   Düğüm_Bölme(s,1,r)
   Düğüm_Ekle_DoluDeğil(s,k)
 değilse
   Düğüm_Ekle_DoluDeğil(r,k)
```

Algoritma 20.5. Düğüm_Ekle_DoluDeğil(x,k)

▷ x düğümüne k kaydı eklenmektedir.

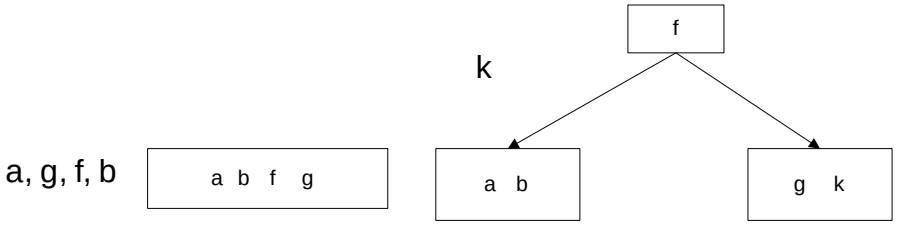
```
i←n[x]
eğer yaprak[x] ise
   i≥1 ve k<anahtari[x] olduğu sürece devam et
      anahtari+1[x] ←anahtari[x]
      i←i+1
   anahtari+1[x]←k
   n[x] ←n[x]+1
değilse
  i≥1 ve k<anahtari[x] olduğu sürece devam et
      i←i-1
   i←i+1
   eğer n[ci[x]]=2t-1 ise
       Düğüm_Bölme(x,i,ci[x])
       eğer k>anahtari[x] ise
          i←i+1
   Düğüm_Ekle_DoluDeğil(ci[x],k)
```

Algoritma 20.5' in çalışma şekli şöyledir. Eğer x bir yaprak düğümse, k kaydı bu düğüme eklenir. Eğer x düğümü yaprak değilse, bu durumda x düğüme bağlı olan bir alt ağacın uygun bir yaprağına eklenir.

Düğüm ekleme işleminde eklenecek kayıt yaprak düğüme eklenir. Bundan dolayı arama işlemi ağacın kök düğümünden başlayıp yapraklara kadar gider. Bundan dolayı Düğüm_Ekleme algoritmasının mertebesi O(h) olur. arama işlemlerinde düğüm içerisinde ardışıl arama yapıldığından ağaca düğüm ekleme işleminin mertebesi $O(th)=O(t\log_t n)$ olur.

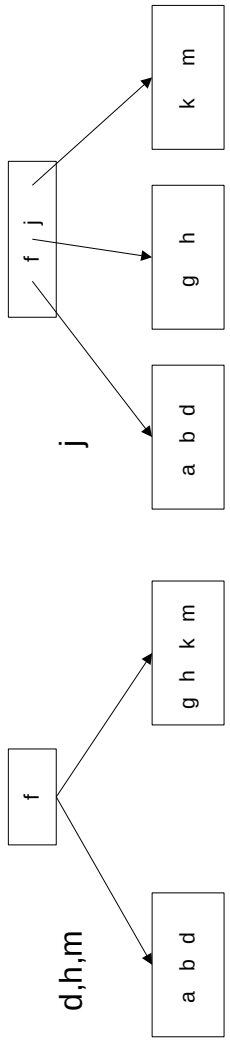
Örnek 20.1

Mertebesi 5 olan bir B-ağacına a, g, f, b, k, d, h, m, j, e, s, i, r, x, c, l, n, t, u, p anahtarları bu ağaca eklenmek isteniyor.



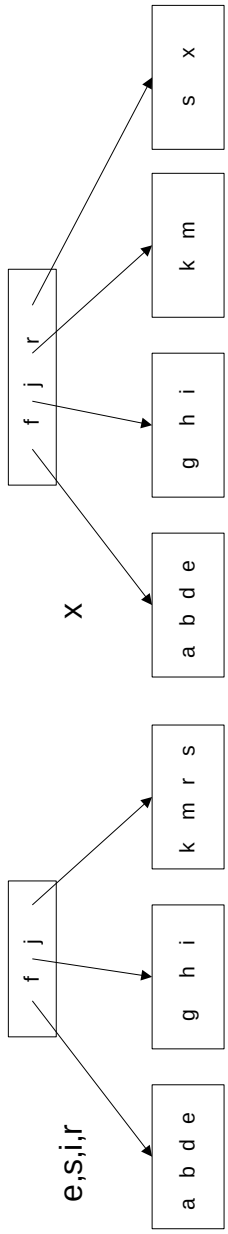
Şekil 20.5. *a, g, f, b, k anahtarlarının mertebesi 5 olan T B-ağacına eklenmesi.*

Şekil 20.5' te ilk olarak a, g, f, b anahtarlarının eklenmesi görülmektedir. Bu anahtarlar bir düğüme sığdıklarından dolayı eklenme işlemi basitçe gerçekleştirilmiştir. Fakat k anahtarı eklenirken eklenecek düğüm dolu olduğundan düğüm bölünüyor ve ağacın seviyesi 1 artıyor.

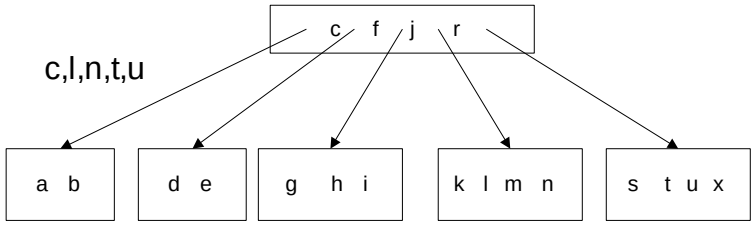


Şekil 20.6. *T ağacına d, h, m, j anahtarlarının eklenmesi.*

Şekil 20.6’ da d, h, m, j anahtarlarının T ağacına eklenmesi görülmektedir. d, h, m değerlerinin eklenmesi sırasında düğümlerde boş yer olduğundan ekleme işlemi kolayca gerçekleştirilir. Fakat j kaydı ekleneceği zaman denk geldiği düğümde boş yer olmadığından düğüm parçalanması meydana gelir ve ağaçta bir düğüm artar.

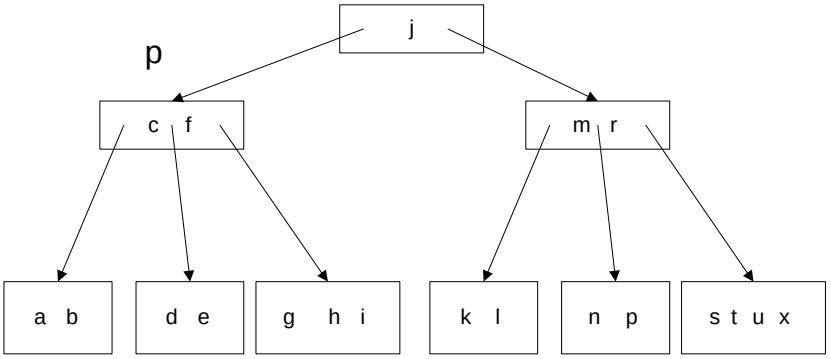


Şekil 20.7. *T ağacına e,s,i,r,x anahtarlarının eklenmesi.*



Şekil 20.8. *T ağacına c, l, n, t, u anahtarlarının eklenmesi.*

Şekil 20.7 ve Şekil 20.8' de T ağacına e, s, i, r, x, c, l, n, t ve u anahtarlarının eklenmesi görülmektedir. Bu anahtarlardan sadece x ağacın şekil değişikliğine uğramasına sebep olmuştur. Son olarak p anahtarı ağaca eklendiğinde ağacın şekli değişiyor ve ağacın seviyesi bir artıyor.



Şekil 20.9. *T ağacına p anahtarının eklenmesi ve ağacın yapısında meydana gelen değişiklik.*

B-ağaçlarında, anahtar silme işlemi, anahtar ekleme işleminin tersi olarak gerçekleşir. Ekleme işleminde anahtar, ilk olarak yaprağa gider ve ondan sonra gerekiyorsa, ağaçta değişiklik yapılır. Silme işleminde de ilk olarak yapraktan anahtar silinmesi istenir. Eğer silinecek anahtar yaprakta değilse, bu anahtarın bir öncesi (veya sonrası) doğal sıralama altında yaprakta olacağı garantidir. Bu durumda yapraktaki bir öncesi (veya sonrası) yapraktan silinecek anahtarın olduğu yere kaydırılır ve yapraktan silinir.

Algoritma 20.6. Düğüm_Silme(T,k)

▷ T ağacından k anahtarı silinmek isteniyor.

```
ÖzYinelemeli_Sil(T,k,Bulundu)
eğer Bulundu=0 ise
   hata
değil ve n[T]=0 ise
    boş kök düğümünü sil
```

Algoritma 20.7. ÖzYinelemeli_Sil(T,k,Bulundu)

```
eğer T=NIL ise
  Bulundu←0
değilse
   DüğümAra(T,k,Bulundu,z)
   eğer Bulundu=1 ise
      eğer c_{z-1}[T]=NIL ise
         Sil(T,z)
      değilse
         Sonraki(T,z)
         ÖzYinelemeli_Sil(c_z[T],  anahtar_z[T], Bulundu)
         eğer Bulundu=0 ise
           Hata
   değilse
      ÖzYinelemeli_Sil(c_z[T], k, Bulundu)
      eğer c_z[T]≠NIL ise
         eğer n[c_z[T]]<t-1 ise
            TekrarYükle(T,z)
```

Eğer yaprakta minimum anahtardan daha fazla anahtar varsa, problem çıkmaz ve basitçe yukarı kaydırılan anahtar silinir. Eğer yaprak minimum sayıda anahtar içermekteyse, bu durumda aynı yaprakla ortak ataya sahip olan sağdaki ve soldaki komşu yapraklara bakılır. Eğer komşu düğümlerden biri minimumdan fazla anahtara sahipse, o yapraktan bir anahtar ata düğüme kaydırılır ve ata düğümden bir anahtar silinecek kaydın üzerine kaydırılır. Eğer bitişik yapraklar minimum sayıda anahtara sahiplerse, iki tane bitişik yaprak ile atadaki medyanları birleştirilerek bir yaprak oluşturulur. Bu adımdan sonra eğer ata düğümde minimumdan daha az anahtar kalırsa, aynı işlem yukarıya doğru uygulanır. Son basamak olarak kök düğümdeki tek kayıt silinirse, ağacın yüksekliği bir azalır. Algoritma 20.6, Algoritma 20.7, Algoritma 20.8, Algoritma 20.9, Algoritma 20.10, Algoritma 20.11, Algoritma 20.12, Algoritma 20.13 Algoritma 20.14 algoritmaları hepsi bir arada kullanılarak T gibi bir ağaçta kayıt silme işlemini gerçekleştirirler.

Algoritma 20.8. Sil(T,z)

```
i←z+1 ... 2t-1 kadar
  anahtar_{i-1}[T]←anahtar_i[T]
  c_{i-1}[T]←c_i[T]
n[T] ←n[T]-1
```

Algoritma 20.9. Sonraki(T,z)

```
q←c_z[T]
c_1[q]≠NIL olduğu sürece devam et
  q←c_1[q]
anahtar_z[T]←anahtar_1[q]
```

Algoritma 20.10. TekrarYükle(T,z)

```
eğer k=0 ise
  eğer n[c1[T]]>t-1 ise
     SolaHareket(T,1)
   değilse
     Birleştir(T,1)
değil ve eğer z=2t-1 ise
   eğer n[cz-1[T]]>t-1 ise
     SağaHareket(T,z)
   değilse
     Birleştir(T,z)
 değilse
   eğer n[cz-1[T]]>t-1 ise
     SağaHareket(T,z)
   değil ve eğer n[cz+1[T]]>t-1 ise
     SolaHareket(T,z+1)
   değilse
     Birleştir(T,z)
```

Algoritma 20.11. DüğümAra(T,k,Bulundu,z)

```
eğer k<anahtar1[T] ise
   Bulundu←0
   z←0
değilse
   z←n[T]
   k<anahtarz[T] ve z>1 olduğu sürece tekrar et
     z←z-1
   Bulundu←(k=anahtarz[T])
```

Algoritma 20.12. SağaHareket(T,z)

```
q←c_z[T]
i←n[q] ... 1 kadar
    anahtar_{i+1}[q]←anahtar_i[q]
    c_{i+1}[q]←c_i[q]
c_2[q]←c_1[q]
n[q] ←n[q]+1
anahtar_1[q]←anahtar_z[q]
q←c_{z-1}[T]
anahtar_z[T]←anahtar_{n[q]}[q]
c_1[c_z[q]]←c_{n[q]}[q]
n[q] ←n[q]-1
```

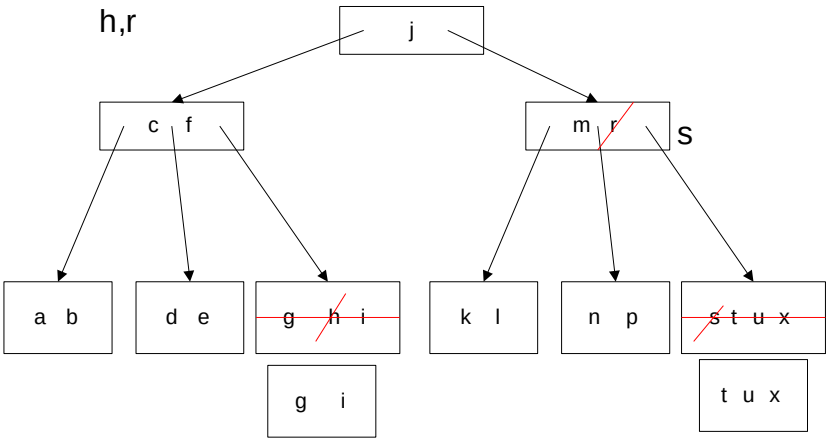
Algoritma 20.13. SolaHareket(T,z)

```
q←c_{z-1}[T]
n[q] ←n[q]+1
anahtar_{n[q]}[q]←anahtar_z[q]
c_{n[q]}[q]←c_1[c_z[q]]
q←c_z[T]
anahtar_z[q]←anahtar_1[q]
c_1[q]←c_2[q]
n[q] ←n[q]-1
i←1 ... n[q] kadar
    anahtar_i[q]←anahtar_{i+1}[q]
    c_i[q]←c_{i+1}[q]
```

Algoritma 20.14. Birleştir(T,z)

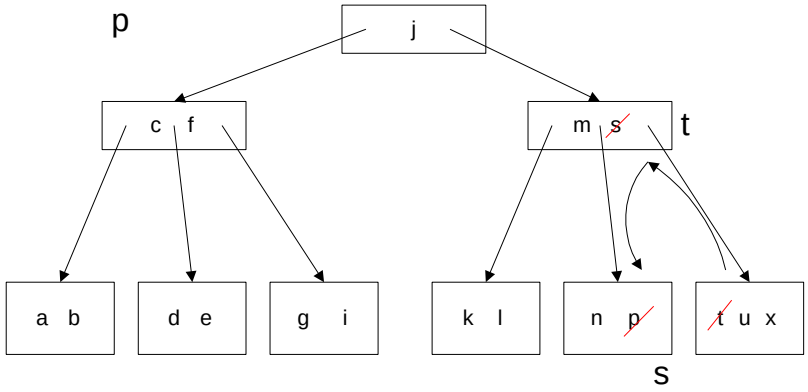
$q \leftarrow c_z[T]$
$p \leftarrow c_{z-1}[T]$
$anahtar_{n[p]}[p] \leftarrow anahtar_z[p]$
$c_{n[p]}[p] \leftarrow c_1[q]$
$i \leftarrow 1 \ldots n[q]$ kadar
  $n[p] \leftarrow n[p]+1$
  $anahtar_{n[p]}[p] \leftarrow anahtar_i[q]$
  $c_{n[p]}[p] \leftarrow c_i[q]$
$i \leftarrow 1 \ldots n[T]-1$ kadar
  $anahtar_i[T] \leftarrow anahtar_{i+1}[T]$
  $c_i[T] \leftarrow c_{i+1}[T]$
$n[T] \leftarrow n[T]-1$

Şekil 20.9 verilen B-ağacı üzerinde silme işleminin nasıl çalıştığı gösterilebilir. İlk olarak h ve r anahtarlarının silineceği kabul edilsin. h anahtarı kolaylıkla silinir. Fakat r anahtarı kolaylıkla silinemez, çünkü bulunduğu düğüm minimum sayıda anahtara sahiptir. Bundan dolayı sağ çocuğundan s değeri yukarı kaydırılır ve çocuktan silinir. Ondan sonra bu s anahtarı r yerine eklenir.



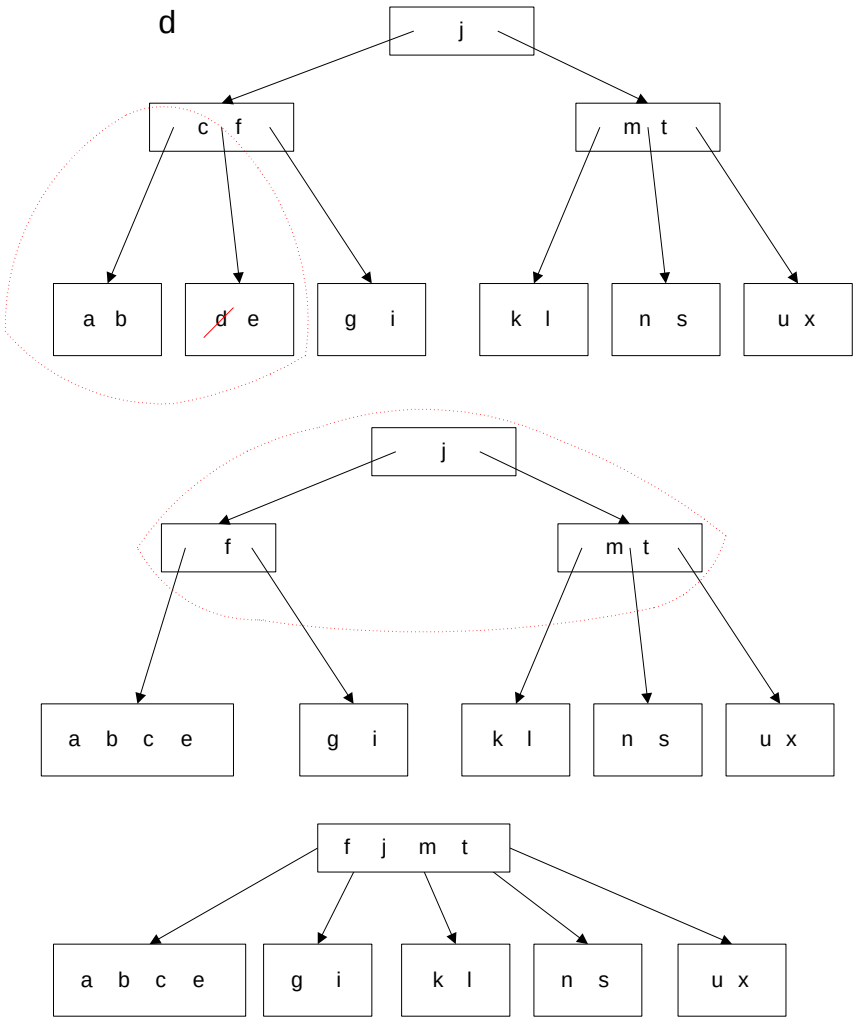
Şekil 20.10. *h ve r anahtarlarının Şekil 20.9' daki T ağacından silinmesi.*

p anahtarı silinmek istendiğinde işlemler kolayca bitirilemez. Çünkü p anahtarı bulunduğu yapraktaki anahtar sayısı minimumdur ve bu yaprağın atası olan düğümde de minimum sayıda anahtar olduğundan, bitişik yapraklardan birinde destek alınacaktır. Sağdaki yaprakta fazladan anahtar olduğundan t ata düğüme kaydırılırken s anahtarı yaprağa kaydırılır ve p anahtarı silinir.



Şekil 20.11. *p anahtarının ağaçtan silinmesi.*

Son olarak d anahtarı silinmek istendiği zaman, d anahtarının bulunduğu yaprak minimum anahtara sahip ve komşu yapraklar da minimum anahtara sahiptirler. Aynı zamanda ata düğümde minimum düğüme sahiptir. Bundan dolayı Şekil 20.12' de görüldüğü gibi ağacın seviyesi 1 azalmaktadır.



Şekil 20.12. *d anahtarının T ağacından silinmesi.*

Bölüm Soruları

20.1. Şekil 20.9' daki ağaca z, v, o, q, w, y anahtarlarını ekleyiniz.

20.2. Mertebesi sırası ile 3, 4 ve 7 olan boş B-ağaçlarına a g f b k d h m j e s i r x c l n t u p anahtarlarını ekleyiniz.

20.3. Mertebesi 5 ve yüksekliği 3 olan bir B-ağacında bir tane anahtarın silinmesi ağaçta seviye azalması olması için bu ağaçta kaç tane anahtar olmalıdır?

20.4. B-ağaçlarında mertebenin 1 olmasına izin verilmez. Neden?

20.5. Bir B-ağacında h yüksekliği, t minimum dereceyi temsil etsin. Bu ağaçta saklanabilecek anahtar sayısı için sıkı bir üst sınır değeri belirleyiniz.

20.6. Bir B-ağacında minimum anahtara nasıl ulaşılır ve verilen bir anahtarın öncesine nasıl ulaşılır? Açıklayınız.

20.7. {1,2,....,n} kümesinin elemanları minimum derecesi 2 olan bir B-ağacına eklenmektedir. Bu kümenin bütün elemanları eklendikten sonra ağaçta kaç tane düğüm olur ?

20.8. B-ağacında arama işlemi yapılırken düğümler içerisinde ardışıl arama yapıldığı daha önce vurgulanmıştı. Eğer düğümler içerisinde ardışıl arama yerine ikili arama yapılırsa, B-ağaçları üzerinde arama işleminin mertebesi ne olur?