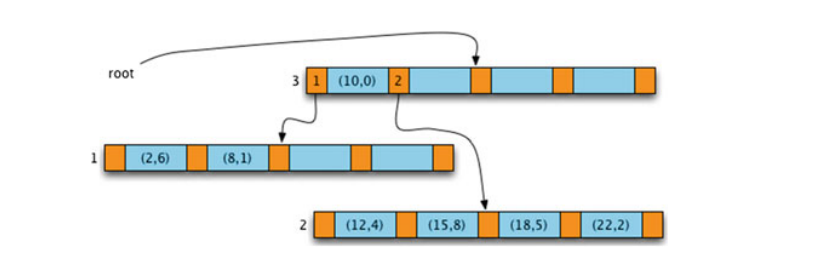
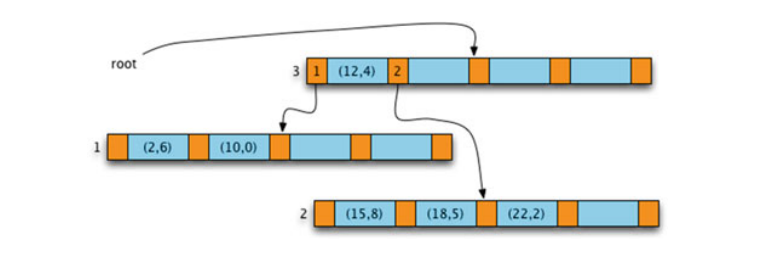


**Fig. 11.12** 14**‘**ü İçeren Öğeyi Sildikten Sonra



**Fig. 11.13** 50’yi İçeren Öğeyi Sildikten Sonra

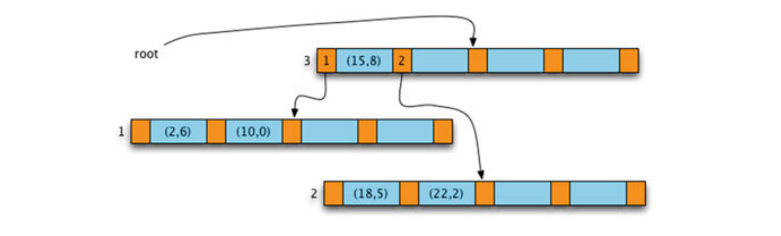


**Fig. 11.14** 8’i İçeren Öğeyi Sildikten Sonra

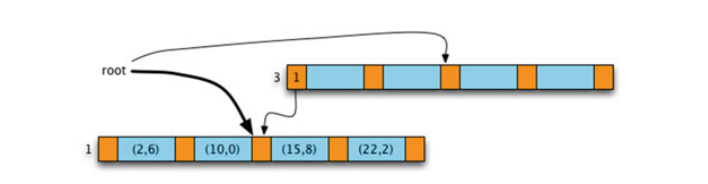
14'ü içeren düğüm dengesiz hale gelir. Dengeleme, sol kardeşinden öğeler ödünç alınarak yapılır. Bu, Şekil 11.12'de gösterilmiştir. Şekil 11.12'de, 10'un ebeveyne döndüğünü ve 12'yi içeren öğenin ağacın 2. düğümüne döndüğünü fark edin. Bu, düğümler arasındaki sıralamanın korunması için gereklidir. Dönme işlemi, öğelerin iki düğüm arasında yeniden dağıtılması için ebeveyne doğru hareket eder. Şimdi, 50'yi içeren öğenin silinmesini düşünün. Bu durumda sağda hiçbir kardeş yoktur ve soldaki kardeşin öğeleri onları yeniden dağıtacak kadar yeterli değildir. Bu nedenle, 2. ve 4. düğümler, kök düğümdeki 18 öğesiyle birlikte tek bir düğümde birleştirilir ve Şekil 11.13'te gösterilen B-Tree ortaya çıkar.

Sonraki adımda, 8 B-Tree silinir. Bu, sağdaki kardeşiyle birlikte bir sola dönüşüm (left rotation) yapılmasına neden olur ve bu da Şekil 11.14'te gösterilen B-Tree oluşturur.

Örneğe devam edersek, 12 anahtarını içeren öğenin ağaçtan silindiğini varsayalım. Bu öğe, yaprak olmayan bir düğümdedir, bu nedenle bu durumda sağ alt ağacın en küçük değeri, 12'yi içeren öğeyi yerine koyar. Bu işlem, ardından silme işlemiyle devam etmelidir.



**Fig. 11.15** 12'yi İçeren Öğeyi Sildikten Sonra



**Fig. 11.16** 18’i içeren öğeyi sildikten sonra

Bu değer, bu durumda sağ alt ağaçtan 15'i içeren öğedir. Sonuç Şekil 11.15'te gösterilmektedir.

Daha sonra 18'in silinmesi, iki kardeş düğümün ana öğedeki (bu durumda kök) ayrılan öğeyle birlikte kömürleşmesine neden olur. Sonuç, Şekil 11.16'da gösterildiği gibi boş bir kök düğümdür. Bu durumda B-Tree sınıfındaki delete yönteminin bu durumu tanıması ve kök düğüm işaretçisini doğru düğümü işaret edecek şekilde güncellemesi gerekir. B-Tree düğümü 3 artık B-Tree'nin kök düğümü değildir. Düğümlerden herhangi birinin silinmesi, kök düğümdeki öğe sayısını azaltır.

Yine B-Trees düğümlerindeki silme yöntemi yinelemeli olarak uygulanabilir. B-Trees düğüm silme yöntemine silinecek öğe verilir ve herhangi bir şey döndürmesi gerekmez. Özyinelemeli algoritma aşağıdaki şekilde ilerler.

1. Silinecek öğe mevcut düğümdeyse, bunun yaprak düğüm olup olmadığına bağlı olarak iki şeyden birini yaparız.

a. Düğümün bir yaprak düğüm olması durumunda öğe, yeniden dengelemeye bakılmaksızın düğümden silinir.

b. Düğüm yaprak olmayan bir düğümse, sağ alt ağaçtaki en küçük değerli öğe, öğenin yerini alır ve sağ alt ağaçtan en küçük değerli öğe silinir.

2. Öğe geçerli düğümde değilse, silme işlemi doğru alt ağaçta özyinelemeli olarak çağrılır

3. Silme geri döndükten sonra, çocuğun silinen öğeye giden yolda yeniden dengelenmesi gerekebilir. Alt düğümün dengesi bozuksa önce soldan bir değer döndürmeyi deneyin B-Tree Delete veya sağ kardeş. Bu yapılamıyorsa alt düğümü sol veya sağ kardeşle birleştirin.

Algoritma özyinelemeli olarak yerine yinelemeli olarak uygulanırsa, kök düğümden silinecek öğeyi içeren düğüme giden yolu takip etmek için bir yığına ihtiyaç vardır. Öğeyi sildikten sonra yığın boşaltılır ve her düğüm yığından çıkarıldığında, yukarıdaki adımlarda açıklandığı gibi yoldaki alt düğümün yeniden dengelenmesi gerekebilir.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**11.8**  **Bölüm Özeti**

B-Trees, özellikle ilişkisel veritabanları için çok önemli veri yapılarıdır. Birleştirme işlemlerinin verimli bir şekilde gerçekleştirilebilmesi için ilişkisel bir veritabanında en azından bazı tablolar üzerinde indekslere ihtiyaç vardır. B-Trees ayrıca disk üzerinde kayıt formatında saklanabildikleri için de önemlidir, yani tüm dizinin herhangi bir anda RAM'de bulunması gerekmez. Bu, milyonlarca kayıttan oluşan tablolar için bile B-Trees oluşturulabileceği anlamına gelir.

B-Ağaçları, O(log n) arama, ekleme ve silme süresi dahil olmak üzere birçok önemli özelliğe sahiptir. B-Ağaçları, ekleme ve silme işlemlerinin sırası ne olursa olsun her zaman dengeli kalır. B-Ağaçları ayrıca bir tablodaki kayıtlara artan veya azalan bir sırayla sıralı erişim sağlayabilir.

B-Trees'teki denge gereksinimi nedeniyle, öğe ekleme sırasında düğümlerin bölünmesi gerekebilir. Öğe silme sırasında düğümlerin yeniden dengelenmesi gerekebilir. Yeniden dengeleme, öğelerin rotasyonu veya düğümlerin birleştirilmesi şeklinde gerçekleşir. Öğeleri yeniden dağıtmak için rotasyon, yeniden dengelemenin tercih edilen yöntemidir.

Hem ekleme hem de silme işlemleri özyinelemeli veya yinelemeli olarak uygulanabilir. Her iki durumda da bölme veya yeniden dengeleme, öğeyi eklemek veya silmek için izlenen yolda etkiler, ağaç boyunca dalgalandıkça basamaklı bölme veya yeniden dengeleme ile sonuçlanabilir. Yinelemeli olarak uygulandığında, hem ekleme hem de silme algoritmaları, kök düğümden eklenen veya silinen öğeye giden yolu kaydetmek için bir yığın gerektirir, böylece bu dalgalanma etkisi ele alınabilir. Özyinelemeli durumda, çalışma zamanı yığını kök düğümden eklenen veya silinen öğeye giden yolu hatırladığından yığına gerek yoktur.

B-Ağaçlarının oluşturulmuş türev uygulamaları vardır. B+-Trees ve B#-Trees bu metinde ele alınmayan diğer iki varyasyondur. Alan Tharp [7], diğerlerinin yanı sıra, bu türev uygulamaların her ikisini de kapsamaktadır.

**11.9 Soruları Gözden Geçirin**

Bu kısa cevaplı, çoktan seçmeli ve doğru/yanlış soruları cevaplayarak bölüme hakimiyetinizi test edin.

1. Bir indeksin kullanılması, Bölüm 11.2.7'de sunulan örnek birleştirme işleminin verimliliğini nasıl artırır?

2. B-Tree'nin anindex'in hash tablo uygulamasına göre ne gibi avantajları vardır?

3. Bir hash tablosunun bir anindex’in B-Tree uygulamasına göre ne gibi avantajları vardır?

4. Milyonlarca kayıt içeren bir tablo üzerinde B-Tree dizini nasıl oluşturulup hala kullanılabilir hale gelebilir? Bu ne gibi zorluklara yol açabilir ve B-Tree bu zorluklarla başa çıkmak için nasıl bir araç sağlayabilir?

5. Şekil 11.13 ile başlayarak 13 anahtarlı bir öğe ekleyin ve ortaya çıkan B-Tree'nin resmini çizin.

6. Şekil 11.10'dan başlayarak 12'yi içeren öğeyi silin ve ortaya çıkan B-Tree'nin resmini çizin.

7. Bir düğüm ne zaman birleşir? Bu ne anlama gelir? Metindeki herhangi bir örnekten farklı, kısa bir örnek verin.

8. Rotasyon bir düğümdeki dengesizliği ne zaman düzeltir? Metindeki herhangi bir örnekten farklı kısa bir örnek veriniz.

9. Ekleme algoritmasını anladığınızı göstermek için 1'den 10'a kadar olan değerleri 4. derecedeki boş bir B-Trees’e ekleyin. Resimler çizin ancak bölme gerektirmeyen resimleri birleştirebilirsiniz. Her bölünmede tamamen yeni bir resim çizdiğinizden emin olun.

10. Her silme işleminden sonra yeniden dengelenen ağacı gösteren önceki inceleme sorusunda oluşturduğunuz ağaçtan 7, 8 ve 9 değerlerini silin.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**11.10 Programlama Sorunları**

Bir B-Tree sınıfı ve bir B-Tree düğüm sınıfı yazın. Bu bölümde açıklanan ekleme ve silme algoritmalarını uygulayın. Bir arama yöntemini de uygulayın.

1. Bölüm 11.2.7'de sunulan birleştirme işlemini verimli bir şekilde çalıştırmak için bu uygulamayı kullanın. Bu algoritmanın çalışması için gereken süreyi Bölüm 11.2.5'teki indekslenmemiş birleştirmenin çalışması için geçen süre ile karşılaştırın. İki yöntemi yinelemeli olarak yazın.

2. B-Tree sınıfını yinelemeli, yinelemeli olmayan ekleme ve silme uygulamalarıyla yazın. Bu durumda, B-Tree sınıfının ekleme ve silme yöntemlerinin mutlaka B-Tree düğümlerinde ekleme ve silme çağrısını yapması gerekmez.

3. Bu bölümdeki örnek tablolar oldukça küçük olduğundan, alıştırma 1 veya 2'yi tamamladıktan sonra, dizin için bir sözlük kullanarak sorgu kodunu yeniden çalıştırın. Sorguyu bu şekilde uygulamak için harcanan süreyi B ağacı uygulamasıyla karşılaştırın. Deney sonuçlarını yorumlayınız.