10.3 AVL Ağaçları 245

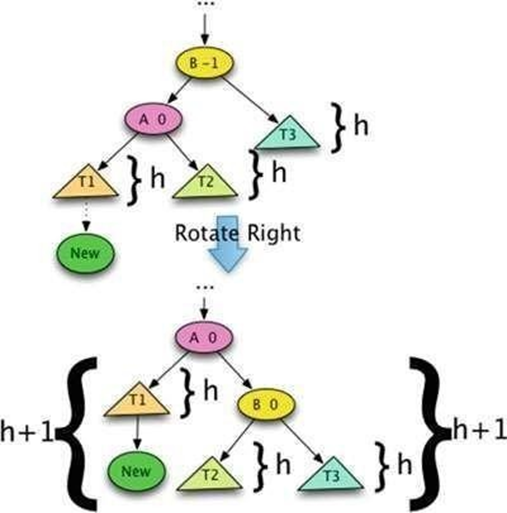
## 10.3.5 Rotasyonlar

Her iki durum 1 ve 2, basitçe dengeleri ayarladıkları için uygulanması önemsizdir. Durum 3, uygulanması en zor olan durumdur. Bir alt ağacı döndürmek, ağaca yeni düğümler eklendikçe ağacın dengede kalmasını sağlayan işlemdir. Durum 3 A için ağaç, ağaca yeni bir düğümün ekleneceği bir durumdadır ve bu da ele alınması gereken bir dengesizliğe neden olur. İki olasılık vardır. Şekil 10.7 bu olası durumlardan ilkini göstermektedir. Yeni düğüm, pivot düğüme bağlı alt ağaç zaten sola ağırlıklıyken, kötü çocuk A'nın soluna eklenebilir. Pivot düğüm olan B, sıfır olmayan bir dengeye sahip en yakın atadır. B düğümünün yeni düğümü eklemeden önce -1 bakiyeye sahip olması için sağ alt ağacının yüksekliği ñ, sol alt ağacının yüksekliği ise *h* + 1 olmalıdır. Yeni düğümün kötü çocuğun alt ağacına eklenmesi, pivotun -2 bakiyeye sahip olmasına neden olur ki buna izin verilmez. Sağa döndürme sorunu çözer ve ikili arama ağacı özelliğini korur. *T2* alt ağacı rotasyonda hareket eder ancak rotasyondan önce *T2*'deki tüm değerler *B'den* küçük *ve A'dan* büyük olmalıdır.

Dengesizlik sağda olduğunda kötü çocuğun sağına bir değer eklemek, sola döndürme gerektiren benzer bir durumla sonuçlanır. Her iki rotasyonda da *A* ve *B* düğümlerinin dengesinin sıfır olduğuna dikkat edin. Bu yalnızca 3 A durumu için geçerlidir ve çift döndürme durumunda geçerli değildir (Şekil 10.8).

Yine, her iki düğümün, pivot ve kötü çocuğun dengesi, her iki yöndeki dönüşten sonra sıfır olur. Durum 3 A başka hiçbir koşul altında mümkün değildir.

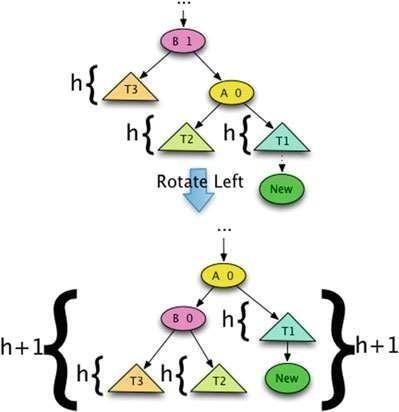
Durum 3B için sadece bir pivot ve kötü çocukla değil, aynı zamanda kötü bir torunla da ilgilenmeliyiz. Önceki bölümde açıklandığı gibi, bu durum bir



Şekil 10.7 AVL Ağacı Vaka 3A Sağa Döndürme

246

10 Dengeli İkili Arama Ağaçları



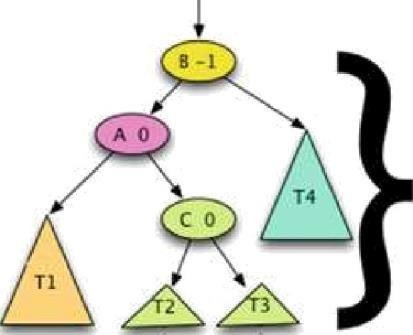
**Şekil 10.8** AVL Ağacı Vaka 3A Sola Döndürme

dengesizliğin tersi yönünde kötü bir çocuğun altına yeni bir değer ekler. Örneğin, Şekil 10.9'daki alt ağaç sola ağırlıklandırılmış ve yeni düğüm kötü çocuğun sağına eklenmiştir. Benzer bir durum, alt ağaç sağa doğru ağırlıklandırıldığında ve yeni düğüm kötü çocuğun sol alt ağacına yerleştirildiğinde meydana gelir. Her iki durum da gerçekleştiğinde, dengeyi yeniden sağlamak için çift döndürme gerekir.

Şekil 10.9 iki olası alt durum olduğunu göstermektedir. Aslında üç olası alt durum vardır. Kötü torun olmaması mümkündür. Bu durumda, yeni eklenen düğüm kötü torun tarafından işgal edilecek olan konuma yerleştirilecektir. Aksi takdirde yeni düğüm, Şekil 10.9'daki C düğümü olan kötü torunun soluna veya sağına yerleştirilebilir. Her iki durumda da Şekil 10.9'daki ilk adım kötü çocuk olan A düğümünü sola döndürmektir. Daha sonra pivot olan B düğümünü sağa döndürerek ağacın yeniden dengelenmesi tamamlanır.

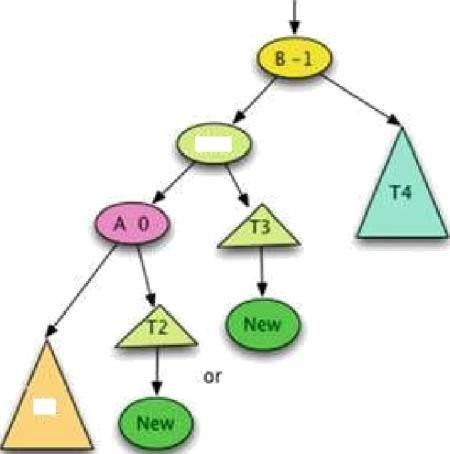
Yine, bu uygulamanın en zor kısmı her bir düğümün bakiyesinin hesaplanmasıdır. Kötü torun ve yenipivot düğüm, Şekil 10.9'daki C düğümü, her zaman 0 bakiyeye sahiptir. Kötü torun yoksa, yeni pivot düğüm yeni eklenen değerdir. Kötü bir torun varsa ve yeni öğe kötü torunun öğesinden daha azsa, kötü çocuğun bakiyesi 0 ve eski pivotun bakiyesi 1'dir. Yeni öğe kötü torunun sağına eklenmişse, kötü çocuğun bakiyesi -1 ve eski pivotun bakiyesi 0'dır. Pivotun üzerindeki bakiyeler de dahil olmak üzere diğer tüm bakiyeler aynı kalır çünkü yeni değer eklenmeden önce ve yeni değer eklendikten sonra ağacın toplam yüksekliği değişmemiştir. Yine, Şekil 10.9'un ayna görüntüsünde benzer bir durum ortaya çıkar. Pivotun sağ alt ağacında bulunan ve halihazırda sağa doğru daha ağırlıklı olan kötü bir çocuğun sol alt ağacına yeni bir değer eklendiğinde, önce kötü çocukta sağa ve sonra pivotta sola dönerek çift döndürme gerekir.

10.3 AVL Ağaçları 247



h+3

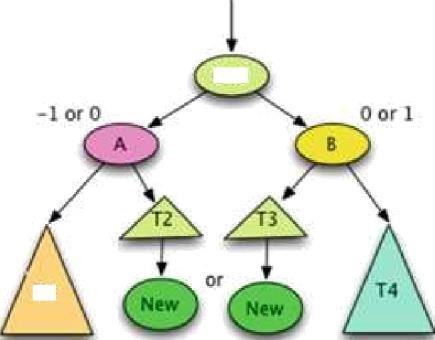
**Adım 1: A'da Sola Döndürün**



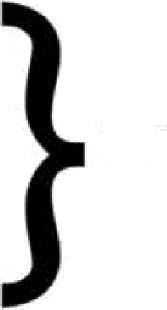
C 0

TI

**Adım 2: B'da Sağa Döndürün**



T}

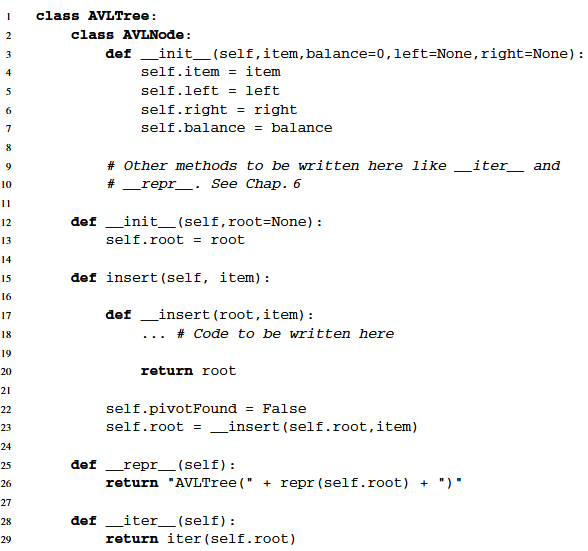


h+3

**Şekil 10.9** AVL Ağacı Vaka 3B Adım 1 ve 2

248 10 Dengeli İkili Arama Ağaçları

**10.3.6 AVL Ağacı Yinelemeli Ekleme**

Özyinelemeli bir işlevi uygularken, bir sınıfın yöntemi yerine tek başına bir işlev olarak yazmak çok daha kolaydır. Bunun nedeni, tek başına bir yöntemin hiçbir şey (yani Python durumunda *None*) üzerinde çağrılabilmesiyken, bir yöntemin her zaman null olmayan bir *self* referansına sahip olması gerektiğidir. Özyinelemeli fonksiyonları metot olarak yazmak *self* için özel durumlara yol açar. Örneğin, *insert* yöntemi özyinelemeli olarak yazılırsa, özyinelemeli işlev olarak \_\_*insert'*i çağırırsa uygulaması daha kolaydır. Bölüm 10.2.1'deki \_\_*insert* fonksiyonu yüksekliği dengeli AVL ağaçları için yeterli olmayacaktır. Ekleme algoritması ağacın mevcut dengesini dikkate almalı ve önceki bölümde sunulan üç durumda tartıştığımız gibi dengeyi korumak için çalışmalıdır.

**1**

**10.3.7 Yinelemeli Ekleme AVL Ağacı Sınıf Bildirimi**

Özyinelemeli uygulamanın kabuğu Bölüm 10.3.7'de verilmiştir. Algoritma, Bölüm 10.2.1'de sunulan insert uygulaması ile birlikte yukarıda sunulan üç durumun bir kombinasyonu gibi ilerler. Özyinelemeli uygulamada *yol yığını* yoktur. Bunun yerine, çalışma zamanı yığını bu amaca hizmet eder. Bölüm 10.2.1'in 5. ve 6. satırları veya 7. ve 6. satırları arasında, kod geri dönüp özyinelemeli çağrılardan yukarı doğru ilerlerken ağacı yeniden dengeleme fırsatı vardır. Her çağrı geri döndüğünde, her düğümün dengeleri buna göre ayarlanabilir.

10.3 AVL Ağaçları 249

Dönmeden önceki bakiyeler, bölümde daha önce açıklandığı gibi birinci ve ikinci durumları uygular. Üçüncü durum, yeniden dengelemeden -2 veya 2 bakiye çıktığında tespit edilir. Bu durumda pivot bulunur ve durum 3'e göre yeniden dengeleme gerçekleşebilir.

Bir pivot bulunması halinde, pivotun üzerinde dengeleme yapılmasına gerek yoktur. Bu, Bölüm 10.3.7'deki kodun 22. satırında ilklendirilen *self.pivotFound* değişkeninin kullanımıdır. Bu bayrak, bulunması halinde pivot düğümün üzerinde herhangi bir dengeleme yapılmasını önlemek için *True* olarak ayarlanabilir. Dengeler, bölümün başlarında vaka bazında analizde açıklandığı gibi ayarlanır. En kötü durumda, pivot ve kötü çocuğun dengelerinin ayarlanması gerekecektir.

AVL ağaçlarına insert'in hem yinelemeli hem de özyinelemeli versiyonlarını uygulamak, yinelemeli versiyonda ele alınması gereken özel durumları göstermeye yardımcı olurken, özyinelemeli versiyon özel durumlara ihtiyaç duymayacaktır. Özyinelemeli versiyon, \_\_*insert'*in çalışma şekli nedeniyle özel durumların ele alınmasına ihtiyaç duymaz. Fonksiyona her zaman yeni öğenin ekleneceği bir ağacın kök düğümü verilir ve bu öğe eklendikten sonra ağacın kök düğümünü döndürür. Bu kadar düzenli bir şekilde çalıştığından, özel durum işleme gerekli değildir.

#### 10.3.8 Yüksekliğe Karşı Dengenin Korunması

Bu bölümde sunulan iki uygulama, AVL ağaçları için özyinelemeli ve yinelemeli ekleme algoritmaları, her bir düğümün dengesini korumuştur. Alternatif olarak, her bir düğümün yüksekliği korunabilir. Bu durumda, bir yaprak düğümün yüksekliği 1'dir. Diğer herhangi bir düğümün yüksekliği 1 artı iki alt ağacının maksimum yüksekliğidir. Boş bir ağacın veya olmamasının yüksekliği 0'dır.

#### 10.3.9 Depolanmış Yükseklikli AVLNode

metin, yazı tipi, ekran görüntüsü, cebir içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Dengeler yerine düğümlerin yüksekliği korunursa, yeni öğenin eklendiği konuma giden yoldaki tüm yükseklikler ağaca geri dönerken ayarlanmalıdır. Dengelerin aksine, pivot düğümde yüksekliklerin ayarlanmasını durdurmak mümkün değildir. Döndürmeden sonra pivot ve kötü çocuğun yüksekliği de yeniden hesaplanmalıdır çünkü döndürme onların yüksekliğini değiştirebilir. Yükseklikler aşağıdan yukarıya doğru hesaplandığından, pivot ve kötü çocuğun yükseklikleri de dahil olmak üzere yol üzerindeki tüm yükseklikler aşağıdan yukarıya doğru yeniden hesaplanmalıdır. Bölüm 10.3.9'daki kod, düğümde köklenen ağacın yüksekliğini saklayan bir AVLNode'un kısmi bildirimini sağlar. Bu uygulamada, herhangi bir düğümün dengesi iki alt ağacın yüksekliklerinden hesaplanabilir.