250 10 Dengeli İkili Arama Ağaçlar

#### AVL Ağacından Bir Öğeyi Silme

Bir AVL ağacından bir değerin silinmesi, Bölüm 6'daki programlama problemi 2'de açıklandığı gibi gerçekleştirilebilir. Ancak, son yaprak düğümün silinmesinden geri dönerken dengelerin ayarlanması gerekir. Bu silme işlemi *yinelemeli olarak* uygulanıyorsabir yol yığınıtutularak ya da silme işleminin özyinelemeli bir uygulamasında özyinelemeli çağrılardan dönerken bakiyeleri veya yükseklikleri ayarlayarak yapılabilir.

Her iki durumda da, yol üzerindeki bir düğümün ayarlanmış dengesi 2'ye ulaştığında, ağacı yeniden dengelemek için sola döndürme gerekir. Yol üzerindeki bir düğümün ayarlanmış dengesi -2 ile sonuçlanırsa, sağa döndürme gerekir. Bu rotasyonlar yol boyunca ağacın köküne kadar kademeli olarak ilerleyebilir.

### Yayvan Ağaçlar

AVL ağaçları her zaman dengelidir, çünkü her düğümün dengesi -1, 1 veya 0 olacak şekilde hesaplanır ve korunur. Dengeli oldukları için *O(log n)* arama, ekleme ve silme süresini garanti ederler. AVL ağacı ikili bir arama ağacıdır, bu nedenle öğelerini sıralı olarak tutar ve *O(n)* sürede en küçük öğeden en büyük öğeye doğru yinelemeye olanak tanır. Bu veri yapısının çok fazla dezavantajı yok gibi görünse de, splay ağaçları şeklinde olası bir iyileştirme vardır.

AVL ağaçlarına yöneltilen eleştirilerden biri de her düğümün kendi dengesini koruması gerektiğidir. Bu denge bakımı için gereken ekstra iş ve ekstra alan gereksiz olabilir. Peki ya bir ikili arama ağacı, dengesini her düğümde saklamadan da *yeterince iyi* koruyabilseydi? Her bir düğümün dengesini ya da yüksekliğini saklamak bellekteki verinin boyutunu artırır. Bellek boyutları daha küçükken bu daha büyük bir endişeydi. Ancak, fazladan bilgiyi korumak da fazladan zaman alır. Peki ya sadece toplam veri boyutunu azaltmakla kalmayıp ikili arama ağacının dengesini koruma işinin bir kısmını ortadan kaldırabilseydik?

AVL ağaçlarında yapılan iyileştirme, *uzamsal yerellik* kavramını içermektedir. Bu fikir, büyük veri setleriyle etkileşimin doğasını yansıtmaktadır. Büyük bir veri setine erişim genellikle yereldir, yani aynı veya birkaç veri parçasına kısa bir süre içinde birkaç kez erişilebilir ve daha sonra yeni değerler ekleyerek veya eski değerlere bakarak verilerin nispeten küçük bir alt kümesine erişilirken bir süre erişilmeyebilir. *Mekansal Yerellik,* verilerin nispeten küçük bir alt kümesine kısa bir süre içinde erişildiği anlamına gelir.

Bu bölümün başındaki örneğimiz açısından, çerezleri içeren bir ağaç, bir kullanıcı bir web sitesini ilk ziyaret ettiğinde atanan çerezlere sahip olabilir. Web sitesine gelen bir kullanıcı bir süre etkileşimde bulunacak ve daha sonra muhtemelen bir daha geri gelmemek üzere ayrılacaktır. Web sunucusuyla etkileşime giren kullanıcı kümesi zaman içinde değişecektir, ancak ağaçtaki toplam giriş sayısına kıyasla her zaman nispeten küçük bir alt kümedir. Son kullanıcıların çerezlerini ağacın tepesine daha yakın bir yerde saklayabilseydik, ağaca yeni bir değer ekleme ve arama süresini iyileştirebilirdik. Karmaşıklık artmayacaktır. Bir öğe eklemek hala

10.4 Yayvan Ağaçlar

251

*O(log n)* zaman alır. Ancak bir öğeyi eklemek veya aramak için gereken toplam süre biraz iyileşebilir. Bu, bir yayılma ağacı için motivasyondur.

Bir yayma ağacında, her ekleme veya arama, eklenen veya bakılan değeri *yayma* adı verilen bir işlemle ağacın köküne taşır. Bir değer silinirken, üst değer ağacın köküne kaydırılabilir. Bir yayma ağacı hala bir ikili arama ağacıdır. Splay ağaçları genellikle dengeli kalır ancak AVL ağacının aksine, bir splay ağacı herhangi bir denge veya yükseklik bilgisi içermez. Bir düğümü köke yaymak, AVL ağaçlarının dönüşlerine çok benzeyen bir dizi dönüşü içerir, ancak küçük bir farkla.

Yayvan ağaçları verilerdeki *mekansal yerellikten* faydalanmak üzere tasarlanmış olsa da, iyi performans göstermek için mekansal yerellikten bağımsız olduklarını belirtmek ilginçtir. Yayvan ağaçları, tamamen rastgele veri setlerinde pratikte AVL ağaçları kadar iyi veya daha iyi çalışır.

Yayvan ağaçlarla ilgili ilginç olan birkaç şey vardır.

* İlk olarak, yayma işlemi alt ağaçların yüksekliği hakkında denge veya başka

herhangi bir bilgi gerektirmez. İkili arama ağacı yapısı yeterince iyidir.

* Yayvan ağaçlar her zaman mükemmel şekilde dengeli kalmaz. Ancak nispeten dengeli kaldıkları için ekleme, arama ve silme işlemleri için *O(log n)* ortalama durum karmaşıklığı elde etmeye yetecek kadar dengelidirler. Yeterince iyi oldukları fikri, Bölüm 2'nin ilerleyen kısımlarında ve bu bölümün ilerleyen kısımlarında tartışılan *amorti edilmiş karmaşıklık olarak* adlandırılan şeyin temelini oluşturur.
* Splaying'in uygulanması nispeten basittir.

Bu metinde iki aşağıdan yukarıya yayılan ağaç uygulamasını ele alıyoruz. Yayvan ağaçlar yinelemeli ya da özyinelemeli olarak uygulanabilir ve biz her iki uygulamayı da inceleyeceğiz. Bölüm 6'da ikili arama ağacı ekleme özyinelemeli olarak uygulanmıştır. Eğer yayma özyinelemeli olarak yapılacaksa, yayma insert fonksiyonunun bir parçası olabilir. Yinelemeli olarak yazılırsa, yayma işleminde bir yığın kullanılabilir. İlerleyen bölümler hem yinelemeli hem de özyinelemeli uygulamaları kapsamaktadır. Ancak önce splaying işleminde kullanılan rotasyonları inceleyeceğiz.

#### Yaylanma Dönüşleri

Bir değer her eklendiğinde veya arandığında, bu değeri içeren düğüm bir dizi döndürme işlemi ile en üste yayılır. AVL ağaçlarından farklı olarak, bir splay ağacı bir düğümü, eğer bir büyükbaba varsa, büyükbabasının seviyesine taşımak için çift döndürme kullanır. Bir dizi çift döndürme işlemi sayesinde düğüm ya köke ya da kökün çocuğuna ulaşacaktır. Eğer yayılan düğüm kökün çocuğuna ulaşırsa, onu köke getirmek için tek bir döndürme kullanılır.

Tekli döndürme fonksiyonları genellikle *zig* veya *zag olarak* etiketlenirken, çiftli döndürmeler yayılan düğümün hareket yönüne bağlı olarak *zig-zig* veya *zig- zag* işlemleri olarak adlandırılır. Bazen düğüm zig-zag hareketi ile hareket ederken bazen de zig-zig hareketi ile hareket eder.

Yayılma, bir değer bir yayılma ağacına eklendiğinde, arandığında veya silindiğinde gerçekleşir. Bir değer arandığında ya aranan değer en üste yayılır ya

252 10 Dengeli İkili Arama Ağaçlar

değer ağaçta bulunmazsa değerin ebeveyni olabilir. Ağaçtan silme işlemi, Bölüm 6'daki problem 2'de açıklandığı gibi diğer ikili arama ağaçlarından silme işlemi gibi uygulanabilir. İkili arama ağacından bir değer silindiğinde, silinen düğümün ebeveyni ağacın köküne yayılır.

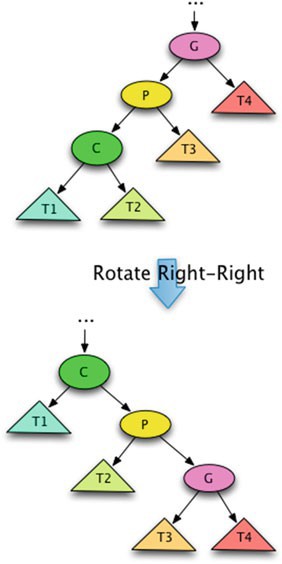
Şekil 10.14'teki örnek, yeşil düğümlerin bir yayma ağacına yerleştirilmesinden kaynaklanan yayma işlemlerini göstermektedir. 30 eklendiğinde, ağacın ikinci versiyonunda (kırmızı düğümler) görüldüğü gibi ağacın köküne doğru yayılır. 5 eklendiğinde, o da köke doğru yayılır. 5'in köke taşınması, çift sağa döndürme adı verilen bir zig-zig döndürme ile gerçekleştirilir. 8'in köke yayılması, sağa-sola döndürme adı verilen bir zig-zag döndürmenin sonucudur. 42 köke doğru yayıldığında, çift sol dönüş ve ardından tek bir sol dönüş gerçekleşir.

15'in köke doğru yayılması, çift sağ dönüş ve ardından sol-sağ dönüş yapılarak gerçekleştirilir. Çift sağa döndürme genellikle çift sola döndürme gibi zig-zig döndürme olarak adlandırılır. Sol-sağ ve sağ-sol rotasyonlar genellikle zig-zag rotasyonlar olarak adlandırılır. Her durumda sonuç, yeni eklenen düğümün veya bakılan düğümün ağacın köküne doğru yayılmasıdır.

Şekil 10.10, 10.11, 10.12 ve 10.13 bu yayılma işlemlerini göstermektedir. Şekil

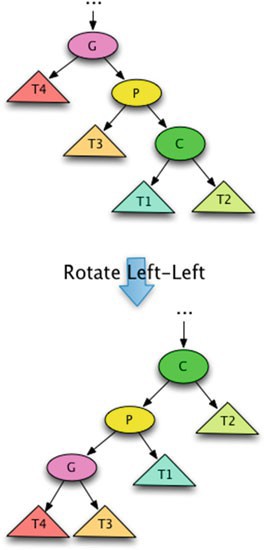
10.12 ve 10.13, yayılma ağaçlarının neden bu kadar iyi çalıştığına dair sezgisel bir

anlayış sunmaktadır

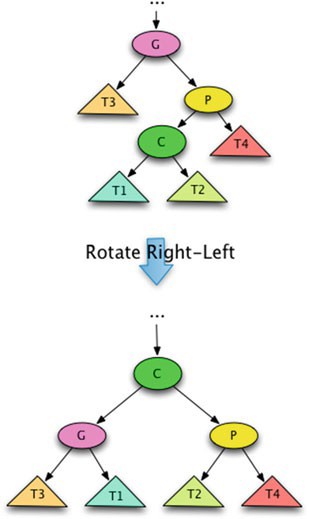


**Şekil 10.10** Yayvan Ağaç Çift-Sağ Döndürme

* 1. Yayvan Ağaçları 253

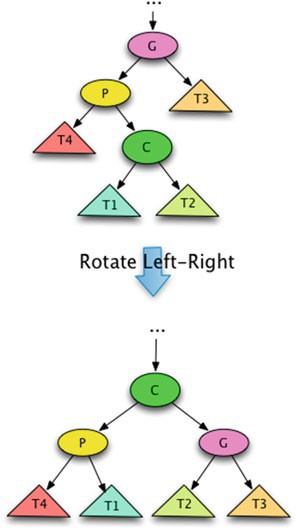


**Şekil 10.11** Yayvan Ağaç Çift-Sol Döndürme



**Şekil 10.12** Yayvan Ağaç Sağ-Sol Döndürme

254 10 Dengeli İkili Arama Ağaçlar



**Şekil 10.13** Yayvan Ağaç Sol-Sağ Döndürme

yapın. Şekil 10.12 ve 10.13'te gösterilen döndürme işlemlerinden sonra, çocukta köklenen alt ağaç bu döndürmelerden öncesine göre daha dengeli görünür.

Sola-sağa döndürme yapmanın, sola döndürme ve ardından sağa döndürme yapmakla aynı şey olmadığına dikkat edin. Sola-sağa döndürme farklı bir sonuç verir. Aynı şekilde, splay sağ-sol döndürme de sağ ve ardından sol döndürmeden farklı bir sonuç verir. Yayvan zig-zag döndürmeler, ağacı dengelemeye yardımcı olmak için bu şekilde tasarlanmıştır. Şekil 10.12 ve 10.13, döndürmeden önce biraz dengesiz olabilecek ağaçların sağ-sol döndürme veya sol-sağ döndürme ile çok daha iyi bir dengeye getirildiğini göstermektedir.