beyaz, yazı tipi, tasarım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.Dengeli İkili Arama Ağaçları

Bölüm 6'da ikili arama ağaçları bir özyinelemeli ekleme algoritması ile birlikte tanımlanmıştır. İkili arama ağaçlarının tartışılması, bazı durumlarda sorunlara sahip olduklarına işaret etmiştir. İkili arama ağaçları dengesiz hale gelebilir, aslında oldukça sık. Bir ağaç dengesiz olduğunda ekleme, silme ve arama işlemlerinin karmaşıklığı O-(n) kadar kötü olabilir. Dengesiz ikili arama ağaçlarıyla ilgili bu sorun, 1962 yılında iki Sovyet bilgisayar bilimcisi olan G.M. Adelson-Velskii ve

E.M. Landis tarafından yükseklik dengeli AVL ağaçlarının geliştirilmesi için motivasyon kaynağı olmuştur. AVL ağaçları adını bu iki mucitten almıştır. AVL ağaçları hakkındaki makaleleri [1], dengeli ikili arama ağaçlarını korumak için ilk algoritmayı tanımlamıştır.

Dengeli ikili arama ağaçları *O(log n)* ekleme, silme ve arama işlemleri sağlar. Buna ek olarak, dengeli bir ikili arama ağacı öğelerini sıralı düzende tutar. Bir ikili arama ağacının infix çaprazlaması, öğelerini artan sırada verecektir ve bu çaprazlama, ağacın zaten oluşturulmuş olduğu varsayılarak O(n) zamanda gerçekleştirilebilir.

HashSet ve HashMap sınıfları da çok verimli ekleme, silme ve arama işlemleri sağlar, ilgili ikili arama ağacı işlemlerinden daha verimlidir. Yığınlar da O(log n) ekleme ve silme işlemleri sağlar. Ancak ne hash tabloları ne de yığınlar öğelerini sıralı bir dizi olarak tutmaz. Çok sayıda ekleme ve silme işlemi gerçekleştirmek istiyorsanız ve bir dizi üzerinde artan veya azalan sırada, belki de birçok kez yinelemeniz gerekiyorsa, dengeli bir ikili arama ağacı veri yapısı daha uygun olabilir.

# Bölüm Hedefleri

Bu bölümde ikili arama ağaçlarının neden dengesiz olabileceği açıklanmaktadır. Daha sonra iki tip yükseklik dengeli ağacın, AVL ağaçlarının ve yayvan ağaçların çeşitli uygulamalarını tanımlamaya devam etmektedir. Bu bölümün sonunda, yinelemeli veya özyinelemeli olarak uygulanan işlemlerle kendi AVL veya splay ağaç veri türünüzü uygulayabilmelisiniz.

© Springer International Publishing Switzerland 2015  
K.D. Lee ve S. Hubbard, Python ile Veri Yapıları ve Algoritmalar,  
Lisansüstü Konularında Bilgisayar Bilimi, DOI 10.1007/978-3-319-13072-9\_10

237

238

10 Dengeli İkili Arama Ağaçları

# İkili Arama Ağaçları

İkili arama ağacı, bir uygulama tarafından çok sayıda ekleme, silme ve arama işlemi gerektiğinde ve bazen öğeleri artan veya azalan sırada gezmek gerektiğinde kullanışlıdır. Wikipedia gibi geniş bir çevrimiçi materyal setine erişim sağlayan bir web sitesi düşünün. Web sitesinin tasarımcılarının son bir saat içinde web sitesine erişen tüm kullanıcıların kaydını tutmak istediğini düşünün. Web sitesi aşağıdaki gibi çalışabilir.

* Her ziyaretçi web sitesine benzersiz bir çerez ile erişir.
* Bir ziyaretçi siteye eriştiğinde, çerezleri tarih ve saatle birlikte sitenin sunucusundaki bir günlüğe kaydedilir.
* Siteye son iki saat içinde erişmişlerse, çerezleri ve erişim zamanları zaten kaydedilmiş olabilir. Bu durumda, son erişim tarih ve saati güncellenir.
* Her saat başı o anda siteye kimlerin eriştiğine dair bir anlık görüntü oluşturulur.
* Anlık görüntü, benzersiz çerez numaralarının artan sırasına göre oluşturulacaktır.
* Anlık görüntüye göre, bir kullanıcı en az bir saat boyunca aktif olmadıktan sonra, bilgileri web sitesi etkinlik günlüğü kayıtlarından silinir.

Site oldukça büyük olduğundan ve her saat binlerce, hatta on binlerce veya daha fazla kişi siteye eriştiğinden, bu bilgileri tutacak veri yapısının hızlı olması gerekir. Girdileri eklemek, aramak ve silmek hızlı olmalıdır. Ayrıca anlık görüntü almak da hızlı olmalıdır çünkü anlık görüntü alınırken web sitesi tüm istekleri bekletecektir.

Wikipedia gibi bir sitede bir saat içinde gelip giden kullanıcı sayısı genellikle uzun süre kalan kullanıcı sayısından fazlaysa, en az bir saat boyunca etkin olmayan her girdiyi silmek yerine etkinlik günlüğünden ağacı yeniden oluşturmak en verimli yöntem olabilir. Bu durum, sitede hala aktif olan kişi sayısı, günlüğün anlık görüntüsündeki aktif olmayan girişlerin sayısından çok daha azsa geçerli olacaktır. Bu durumda, etkin olmayan kullanıcıları sildikten sonra günlüğü yeniden oluşturmak da hızlı olmalıdır.

İkili arama ağacı, O(log *n)* arama, ekleme ve silme ile O(n) anlık görüntü alma süresini garanti edebiliyorsak, bu günlüğün organizasyonu için mantıklı bir seçimdir. Ancak, ikili arama ağacının büyük bir sorunu vardır. Anlık görüntü alınırken günlüğün yalnızca son aktif kullanıcılarla yeniden oluşturulabileceğini ve ayrıca günlüğün yeniden oluşturulması sırasında çerezlere artan sırada erişileceğini hatırlayın.

Bölüm 10.2.1'de gösterilen ikili arama ağaçları üzerindeki ekleme işlemini düşünün. İkili arama ağacı yeniden oluşturulduğunda, yeni ağaca eklenecek öğeler artan sırada eklenecektir. Sonuç dengesiz bir ağaçtır.

## İkili Arama Ağacı Ekleme

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

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

* 1. İkili Arama Ağaçları 239

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

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Öğeler bir ikili arama ağacına artan sırada eklenir, bunun etkisi yürütmenin her zaman satır 2'den 4, 6, 7 ve 8'e ilerlemesidir. Satır 7'deki sonuç, yeni değeri ikili arama ağacının en sağ konumuna yerleştirir, çünkü o ana kadar eklenen en büyük değerdir. Ortaya çıkan ağaç, aşağı ve sağa doğru uzanan bir çubuktur. Ağaçta herhangi bir denge olmadan, bir sonraki büyük değerin eklenmesi, yeni değerin konumunu bulmak için daha önce eklenmiş olan her bir değerin geçilmesine neden olacaktır. Bu, ilk değerin eklenmesi için sıfır karşılaştırma gerekirken, ikincisinin son konumunu bulmak için bir karşılaştırma gerektirdiği, üçüncü değerin iki karşılaştırma gerektirdiği ve bu şekilde devam ettiği anlamına gelir. Bölüm 2'de kanıtlandığı gibi, ağacı oluşturmak için toplam karşılaştırma sayısı O(n2 )'dir. Bu karmaşıklık, bir saat içinde makul miktarda etkinlik alan herhangi bir site için çok yavaş olacaktır. Buna ek olarak, ikili arama ağacının yüksekliği *n* olduğunda, burada *n* ağaçtaki değer sayısıdır, arama, ekleme ve silme süreleri hem en kötü hem de ortalama durumlar için O*(n)'*dir. Ağaç bir çubuk olduğunda ya da çubuk olmaya yakın olduğunda ikili arama ağacının verimlilik özellikleri bağlı listeden daha iyi değildir.

* 1. **AVL Ağaçları**

Dengeli kalan bir ikili arama ağacı, son bölümde açıklanan web sitesi günlüğünün gerektirdiği her şeyi sağlayacaktır. AVL ağaçları, dengelerini korumak için ek bilgiler içeren ikili arama ağaçlarıdır. Bir AVL ağacının yüksekliğinin O(log *n)* olması garanti edilir, böylece arama, ekleme ve silme işlemlerinin hepsinin O(log n) zamanında tamamlanacağı garanti edilir. Bu garantilerle, bir AVL ağacı *n* öğeden oluşan bir diziden *O(n log n)* zamanda oluşturulabilir. Dahası, AVL ağaçları, ikili arama ağaçları gibi, *O(n)* zamanda artan sırada öğelerini veren bir sıralı geçiş kullanılarak çaprazlanabilir.

## Tanımlar

AVL ağaçlarının nasıl çalıştığını anlamak için birkaç tanım yapmak gerekir.

*Yükseklik(Ağaç):* Bir ağacın yüksekliği bir artı alt ağaçlarının maksimum yüksekliğidir. Bir yaprak düğümün yüksekliği birdir.

*Denge(Ağaç):* İkili bir ağaçtaki bir düğümün dengesi yükseklik(sağ alt ağaç)- yükseklik(sol alt ağaç) şeklindedir.

*AVL Ağacı:* AVL ağacı, ağaçtaki her düğümün dengesinin -1, 0 veya 1 olduğu ikili bir ağaçtır.

240

10 Dengeli İkili Arama Ağaçları

# Uygulama Alternatifleri

Bölüm 6'ya ve ikili arama ağaçlarının uygulanmasına geri dönersek, bir ağaca değer eklemek özyinelemeli olarak yazılabilir. Bir AVL ağacına değer eklemek de özyinelemeli olarak gerçekleştirilebilir. Bir AVL ağacına değer ekleme işlemini bir döngü ve bir yığın kullanarak yinelemeli olarak uygulamak da mümkündür. Bu bölüm her iki alternatifi de incelemektedir.

Ayrıca, bir AVL ağacının dengesi ağaçtaki her bir düğümün yüksekliği ya da ağaçtaki her bir düğümün dengesi kullanılarak korunabilir. AVL ağaç düğümlerinin uygulamaları ya dengelerini ya da yüksekliklerini saklar. Değerler ağaca eklendikçe, etkilenen düğümlerin denge veya yükseklik değerleri ağaca eklenen yeni öğeyi yansıtacak şekilde güncellenir.

# Depolanmış Bakiye ile AVLNode

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

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

İster ekleme işlemi özyinelemeli ister yinelemeli olarak gerçekleştirilsin, Bölüm 6'daki Node sınıfı, düğümün dengesini veya yüksekliğini içerecek şekilde biraz genişletilmelidir.  
Bölüm 10.3.3'teki kod parçasını göz önünde bulundurun. İnceleyeceğimiz ilk AVLTree uygulaması, denge bilgisini saklayan yinelemeli bir algoritmadır. Dikkat edilmesi gereken bir nokta, AVLNode sınıfının, AVLTree sınıfının içinde gizlenmiş olmasıdır. Bu, AVLTree sınıfını kullananların doğrudan AVLNode sınıfına erişmesini engellemek içindir. Python, teknik olarak AVLNode sınıfına dışarıdan erişimi engellemez, ancak konvansiyon gereği kullanıcılar, AVLTree veri yapısının iç yapısına müdahale etmemelidir. AVL ağaçları, bu veri yapısını kullananlar tarafından oluşturulabilir, ancak AVL düğümleri doğrudan oluşturulamaz. Düğümlerin oluşturulması, AVLTree sınıfı tarafından yönetilir.

AVLNode yapıcı fonksiyonu, balance, left ve right değişkenleri için varsayılan değerlere sahiptir. Bu özellik, hata ayıklama sırasında AVL ağaçlarını oluşturmayı kolaylaştırır. \_repr\_ fonksiyonu, AVLNode nesnesini, aynı düğümü yeniden oluşturabilecek bir formatta ekrana yazdırır. Örneğin, print(repr(node)) ifadesi çalıştırıldığında, Python'un örnek bir ağaç oluşturabilmesi için bir düğüm yazdırılır. repr(self.left) ve repr(self.right) ifadeleri, \_repr\_ fonksiyonuna yapılan özyinelemeli çağrılardır, böylece kökü self olan tüm ağaç ekrana yazdırılmış olur. Bölüm. 6'daki \_iter\_ fonksiyonu, AVLTree üzerinde gezinmek için çalışacaktır. Bu yineleyici fonksiyon, ağacın tüm değerlerini artan sırayla döndürür. Bu bölümdeki örnekler, AVL ağacındaki düğümlerin dengesi ile ilgilidir. AVL ağaçlarının yükseklik dengeli olarak doğru bir şekilde uygulanması için düğüm dengesini saklamak yeterlidir. Ancak, düğümün yüksekliğini saklamaktan daha zor olabilir. Bölümün ilerleyen kısımlarında, her düğümün yüksekliğini koruyarak çalışan alternatif algoritmalar ele alınacaktır. AVL ağaçlarında yükseklik veya denge faktörü saklansa da, ağacın çalışma zamanının karmaşıklığı değişmez.

10.3 AVL Ağaçları 241

~~-------------------------------------------------------~~

AVL Ağaçları, ancak belki de ağaçtaki her bir düğümün yüksekliğini korumaktan biraz daha zordur. Bölümün ilerleyen kısımlarında bu algoritmaların her bir düğümün yüksekliğini koruyan modifikasyonları tartışılacaktır. AVL Ağaçlarında ister yükseklik ister denge saklansın, ağaç işlemlerinin karmaşıklığı etkilenmez.