Veri Yapıları ve Algoritmalar 10.Ünite

- Konu:Balanced Binary Search Trees(Ağaçlar)

10.1:Dengeli İkili Arama Ağaçları

10.2:İkili Arama Ağaçları

10.3:AVL Ağaçları

10.4:Yayvan Ağaçlar

10.5:Yinelemeli Yayma(Iterative)

10.6:Yinelemeli Yayma(Recursive)

10.7:Performans Analizi

Bölüm sonu soruları

Anlatacağım kısımlar

- 10.5:Yinelemeli Yayma(Iterative)
- 10.6 Yinelemeli Yayma(Recursive)
- 10.7:Performans Analizi

Ayşe Kolutek 2023481049

Yinelemeli Yayma(Iterative)

Bir değer her eklendiğinde veya arandığında, önceki bölümde açıklandığı gibi bir dizi döndürme işlemiyle splay ağacının köküne yayılır. Çift döndürme işlemleri değeri ya ağacın köküne ya da kökünün çocuğuna taşıyacaktır. Çift döndürme işlemi yeni eklenen değerin ağacın kökünün çocuğunda olmasıyla sonuçlanırsa, Şekil 10.14'te 30 ve 15'in yayma ağacına eklendiğinde gösterildiği gibi yeni eklenen değeri köke taşımak için tek bir döndürme kullanılır.

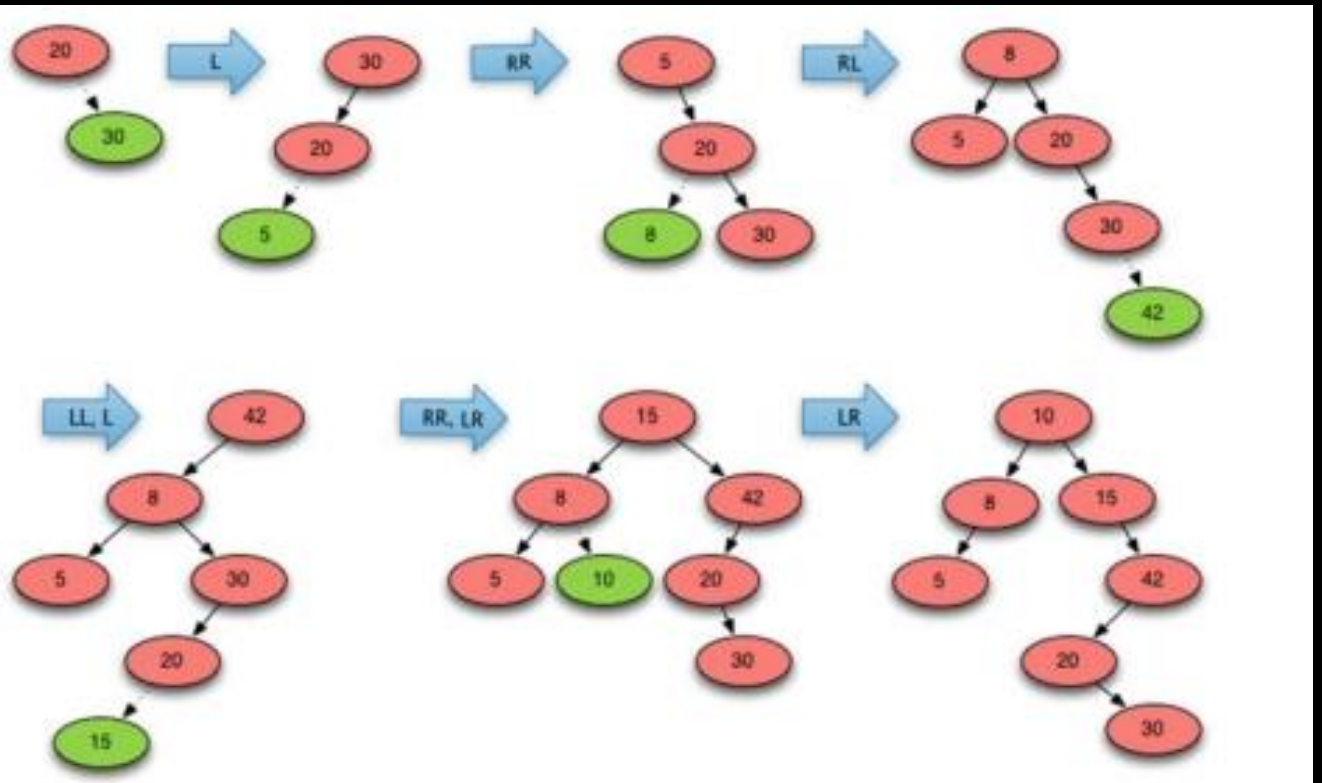


Fig. 10.14 Splay Tree Example

İkili arama ağacına özyineleme olmadan yeni bir değer eklemek bir while döngüsü kullanılarak mümkündür. While döngüsü ağacın kökünden yeni düğümün ebeveyni olacak yaprak düğüme doğru hareket eder, bu noktada döngü sonlanır, yeni düğümoluşturulur ve ebeveyn yeni çocuğuna bağlanır. Yeni düğüm eklendikten sonra, üste doğru yayılması gerekir. Yaymak için ağaç boyunca yeni eklenen düğüme giden yolun bilinmesi gerekir. Bu yol bir yığın kullanılarak kaydedilebilir. Ekleme döngüsü ağaçtaki başka bir düğümden geçerken, bu düğüm yığının üzerine itilir. Sonuç olarak, kökten yeni çocuğa giden yol üzerindeki tüm düğümler bu yol yığınına itilir. Son olarak, bu yol yığını boşaltılarak yayılma gerçekleşebilir. Önce çocuk yığından çıkarılır. Ardından, yığının geri kalanı aşağıdaki gibi boşaltılır.

- Yığında iki düğüm daha varsa bunlar yeni eklenen düğümün ebeveyni ve büyük ebeveyni olur. Bu durumda, yeni döndürülen alt ağacın kökünün yeni eklenen düğüm olmasıyla sonuçlanan bir çift döndürme gerçekleştirilebilir. Hangi çift döndürmenin gerekli olduğu büyük ebeveyn, ebeveyn ve çocuk değerlerinden belirlenebilir.
- Yığında yalnızca bir düğüm kalırsa, bu yeni eklenen düğümün ebeveynidir. Tek bir döndürme yeni eklenen düğümü yayılma ağacının köküne getirecektir.

Burada açıklanan şekilde splay uygulamak, ağaçta bir değer ararken, bulunsa da bulunmasa da iyi çalışır. Bir değer bulunduğunda yol yığınına eklenecektir. Bir değer bulunamadığında, üst öğe en üste yayılmalıdır; bu, aranan değer bulunamadığında doğal olarak gerçekleşir çünkü yayma işlemi gerçekleştirildiğinde üst öğe yol yığınının en üstünde kalacaktır. Bir splay ağacından bir düğümü silmenin bir yöntemi, tıpkı ikili arama ağacında olduğu gibi silme işlemiyle gerçekleştirilir. Silinecek düğümün sıfır veya bir çocuğu varsa, düğümü silmek önemsizdir. Silinecek düğümün iki çocuğu varsa, sağ alt ağacındaki en soldaki değer silinecek düğümdeki değerin yerini alabilir ve en soldaki değer sağ alt ağaçtan silinebilir. Silinen düğümün ebeveyni ağacın tepesine yayılır. Başka bir silme yöntemi, silinen düğümün önce ağacın köküne yayılmasını gerektirir. Ardından sol alt ağacın en sağdaki değeri köke yayılır. Sol alt ağaç yayıldıktan sonra, kök düğümünün sağ alt ağacı boştur ve orijinal sağ alt ağaç buna eklenebilir. Orijinal sol alt ağaç, yeni oluşturulan yayma ağacının kökü olur.

Yinelemeli Yayma(Recursive)

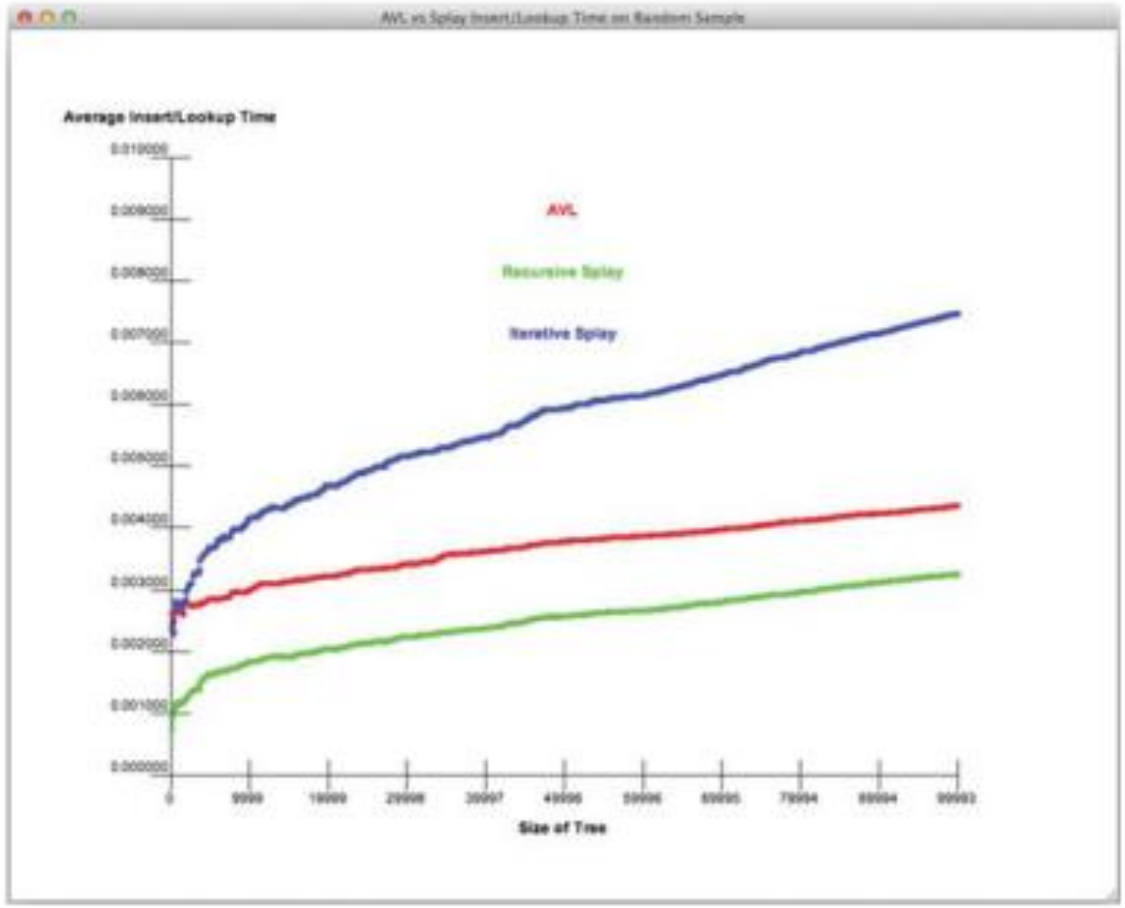
Splaying'in özyinelemeli olarak uygulanması, ikili arama ağaçları üzerindeki özyinelemeli ekleme işlemini takip eder. Splaying bu özyinelemeli insert fonksiyonu ile birleştirilir. Özyinelemeli ekleme ağaçtaki yolu takip ederken "R" ve "L" şeklinde bir döndürme dizesi oluşturur. Yeni öğe mevcut kök düğümün sağına eklenirse, yeni eklenen düğümü ağaçta yukarı doğru yaymak için bir sola döndürme gerekir ve döndürme dizesine bir "L" eklenir. Aksi takdirde, sağa döndürme gerekir ve döndürme dizesine bir "R" eklenir. Özyinelemeli insert fonksiyonu geri döndüğünde, yeni eklenen düğüme giden yol geri dönen fonksiyon tarafından yeniden izlenir. Rotate dizesindeki son iki karakter hangi çift rotasyonun gerekli olduğunu belirler. Bir sözlük veya hash tablosu "RR", "RL", "LR" ve "LL "yi uygun döndürme fonksiyonlarıyla eşleştirir. Hash tablosu araması uygun döndürmeyi çağırmak için kullanılır ve döndürme dizesi kesilir (veya "R" ve "L "nin döndürme dizesine ne zaman eklendiğine bağlı olarak boş dizeye yeniden başlatılır). Özyinelemeli ekleme tamamlandığında, gerekli tekli döndürme rotate dizesine kaydedilir ve gerçekleştirilebilir.

Bu şekilde bir döndürme dizesi ve hash tablosu kullanarak kaydırma uygulamanın, yukarıda açıklanan yinelemeli algoritmaya kıyasla gerekli döndürmeleri belirlemek için koşullu ifadelerin yaklaşık yarısını gerektirdiğine dikkat edilmelidir. Yeni bir düğüm eklerken yol, yol üzerindeki her düğüme eklenecek değerin ağaçtaki konumuyla karşılaştırılmasıyla belirlenmelidir. Yukarıdaki yinelemeli tanımda, yol üzerindeki değerler kaydırma sırasında tekrar karşılaştırılır. Bu özyinelemeli tanımda, yeni öğe yoldaki her öğeyle yalnızca bir kez karşılaştırılır. Bu, bölümün ilerleyen kısımlarında gösterildiği gibi performans üzerinde bir etkiye sahiptir. Bu özyinelemeli uygulamayı kullanarak bir değeri aramak, bulunan değeri ya da bulunamazsa üstünü ağacın köküne yayarak eklemekle benzer şekilde çalışır. Bir değerin silinmesi yine özyinelemeli olarak, önce silinecek değere bakılarak ağacın köküne yayılması ve ardından önceki bölümde açıklanan kök kaldırma yönteminin gerçekleştirilmesiyle yapılabilir.

Performans Analizi

En kötü durumda bir yayvan ağaç her bir arama, ekleme ve silme işlemi için①(n) karmaşıklığa neden olan bir çubuk haline gelebilirken AVL ağaçları arama, ekleme ve silme işlemleri için①(log n) zaman garanti eder. AVL ağaçlarının daha iyi performansa sahip olabileceği görülmektedir. Ancak, pratikte durum böyle görünmemektedir. Önceden oluşturulmuş bir veri kümesi kullanılarak yapılan bir deneyde 100.000'e yakın ekleme ve 900.000 rastgele arama işlemi gerçekleştirilmiştir. Ekleme ve arama işlemleri veri kümesinde tanımlanmış ve aranan tüm değerler ağaçta bulunmuştur. Ortalama birleşik ekleme ve arama süreleri bir AVL ağacı, yinelemeli olarak uygulanan bir splay ağacı ve splay ağacı ekleme ve aramanın özyinelemeli uygulaması için Şekil 10.15'te kaydedilmiştir. Sonuçlar, özyinelemeli splay tree uygulamasının rastgele bir değer kümesinde AVL tree uygulamasından daha iyi performans gösterdiğini ortaya koymaktadır. Deneyler, splay ağaçlarının ekleme ve arama işlemleri için pratikte①(log n) karmaşıklık sergilediğini göstermektedir. Şekil 10.13 ve 10.12'de, yayılan ağaçların özel çift dönüşleri sayesinde dengeyi nasıl koruduğuna dair sezgisel bir anlayış elde ettik. Ancak, çift dönüşlerin ağacı daha dengeli hale getirdiğini söylemek çok ikna edici bir argüman değildir. Bu fikir, amorti edilmiş karmaşıklık kullanılarak resmileştirilmiştir.

İlk olarak Bölüm 2'de karşılaşılan amortisman, bir giderin tamamının bir yıl içinde giderleştirilmesi yerine birkaç yıla yayılması durumunda kullanılan bir muhasebe terimidir. Aynı ilke, bir Splay Tree'de bir değer bulma veya ekleme masrafına da uygulanabilir.bunun tam analizi duruma göre yapılmıştır ve bu metinde mevcut değildir ancak çevrimiçi metinlerde bulunabilir.Bu kanıtlar, splay ağaçlarının rastgele erişilen veriler üzerinde gerçekten de AVL ağaçları kadar verimli çalıştığını göstermektedir. Buna ek olarak, bir değer eklenirken veya aranırken kullanılan yayma işlemi, verideki uzamsal yerellikten faydalanır. Sık sık aranan veri değerleri, gelecekte daha verimli bir şekilde aranmak üzere ağacın tepesine doğru ilerleyecektir. Verilerde mevcutsa uzamsal yerellikten yararlanmak kesinlikle arzu edilir olsa da, yayvan ağaç ekleme ve arama işlemlerinin genel hesaplama karmaşıklığını iyileştirmez. Ancak, bu durum rastgele eklenen ve bakılan değerler üzerinde ortalama durumda gerçekleşmez. Aslında, önceki bölümde sunulan splay ağaçlarının özyinelemeli uygulaması, rastgele dağıtılmış bir değerler kümesinde ①(log n) ortalama ekleme ve arama süresi sergiler ve rastgele bir örneklemde AVL ağacı uygulamasından daha iyi performans gösterir.



Şekil 10.15 Ortalama Ekleme/Arama Süresi

Bir AVL ağacı üzerindeki ekleme, arama ve silme işlemleri ①(log n) zamanda tamamlanabilir. Ortalama durumda bu durum splay ağaçları için de geçerlidir. Bir AVL veya splay ağacının çaprazlanması ①(n) zamanda çalışır ve öğelerini artan veya azalan sırada verir (yineleyicinin nasıl yazıldığına bağlı olarak). Quicksort algoritması bir listenin öğelerini verimli bir şekilde sıralayabilirken, AVL ve splay ağaçları, öğelerinin sıralamasını korurken birçok ekleme ve silme işlemine izin veren veri yapılarıdır. Arama, silme ve ekleme işlemlerini verimli bir şekilde uygulayan ve aynı zamanda değer dizisinin artan veya azalan sırada yinelenmesine izin veren bir veri yapısı gerekiyorsa, AVL veya splay ağacı pratik bir seçim olabilir. AVL ağaçlarının avantajı, verimli arama, ekleme ve silme karmaşıklığını garanti ederken öğelerin sıralamasını koruma yeteneklerinde yatmaktadır. Splay ağaçları neredeyse tüm durumlarda aynı şekilde çalışır ve bu bölümde açıklanan özyinelemeli splay ağacı uygulaması rastgele veri kümelerinde AVL Ağacı uygulamasından bile daha iyi performans gösterir. AVL ağacı ile özyinelemeli splay ağacı performans sayıları arasındaki performans farkı, AVL ağacında dengeyi açıkça korumak ile splay ağacında yeterince iyi denge elde etmek arasındaki farktır.

Bölüm özeti

Bu bölümde yükseklik dengeli AVL ağaçlarının ve yayvan ağaçların çeşitli uygulamaları sunulmuştur. Özyinelemeli ve yinelemeli ekleme algoritmaları sunulmuştur. Hem dengeyi koruyan hem de yüksekliği koruyan AVL düğümleri ele alınmıştır. Hem AVL hem de yayvan ağaçlar için özyinelemeli ekleme algoritmaları, çok fazla özel durum olmadan çok temiz bir kodla sonuçlanırken, yinelemeli sürümler bazı koşulları ele almak için birkaç if deyimine daha ihtiyaç duyar. Bazı durumlarda yinelemeli versiyon özyinelemeli versiyondan biraz daha verimli olabilir, çünkü herhangi bir dilde fonksiyon çağrıları ile ilişkili bir maliyet vardır, ancak bu bölümde yapılan deneylerden elde edilen deneysel sonuçlar, Python'da yazıldığında özyinelemeli uygulamaların çok verimli çalıştığını göstermektedir.

Bölüm sonu soruları

1-AVL ağaç ekleme algoritması neden her zaman ①(log n) zamanda tamamlanır? Bir değerin eklenmesi ile ilgili üç durumun her biri için cevabınızı gerekçelendirmek için durum analizi yapın.

Bir düğüm AVL ağacını seçmeden önce standart bir BST(Binary Search Tree) ekleme işlemi yapılması ve ardından AVL özelliğinin geri kazanılabilmesi için potansiyel olarak ①(log n) döndürme yapılmasını içerir.

BST ekleme işlemi ortalaması olarak ①(log n) zaman karmaşıklığına sahiptir.

Döndürmeler de ortalama olarak ①(log n) zaman karmaşıklığına sahiptir.

Dolayısıyla, AVL ağacına ekleme biriktirme toplam zaman karmaşıklığı ①(log n) olur.

AVL ağaçları, ayrılıklarını dengeleyerek hızlı arama seçenekleri sunar.
Ancak uygulama biraz karmaşıktır ve bir miktar daha fazla döndürme işlemi yapılabilir.

AVL

2- Özyinelemeli ekleme yayma ağacı uygulamasında döndürme dizesinin amacı nedir?

Dengenin sağlanması

Rotasyon işlemleri

Döndürme işlemleri ağacın yapısını değiştirirken dengeli yapısını korur.Döndürme işlemleri sayesinde AVL ağaçları, en kötü durumda bile arama, ekleme ve silme işlemleri O(log n) zaman karmaşıklığında toplanır. Bu, büyük veri kümelerinde önemli bir avantaj sağlar

3-Neden özyinelemeli yayvan ağaç ekleme ve arama uygulaması AVL ağaç uygulamasından daha hızlı çalışıyor gibi görünüyor?

Özyinelemeli yayvan ağaç, arama işlemi sırasında erişilen düğümü köke yakınlaştırmak için yayılan adı verilen bir işlem yapar.

Bu işlem, son erişilen düğümü daha üst seviyelere taşır ve erişimleri hızlandırır.

AVL ağacında ise döndürme işlemleri yapılır, ancak bu işlem yalnızca dengenin korunması için devam eder.

AVL ağacı, her düğümün farklılığını sınırlayarak dengeli bir yapı oluşturur.

Ekleme veya silme işlemi sırasında AVL ağacının dengesi bozulabilir ve döndürme işlemleri yapılabilir.

Özyinelemeli yayvan ağaç ise dengesizlik sistemi daha esnek ele alır ve yayma işlemiyle düğümleri yakınlaştırır.

4-Özyinelemeli ekleme ve arama işlevlerine sahip bir splay ağacı uygulaması yazınız. Her bir düğümün yüksekliğinin korunduğu yinelemeli veya özyinelemeli bir AVL ağacı uygulayın. Ağaçların aynı değerler listesinden oluşturulduğu bir test gerçekleştirin. Değer listesini oluşturduğunuzda, yinelenen değerler bir arama olarak değerlendirilmelidir. Veri dosyasını bir L veya I ile ve ardından bir arama veya ekleme işleminin gerçekleştirilmesi gerektiğini belirten bir değerle yazın. Performans sonuçlarınızı karşılaştırmak için PlotData.py programı tarafından kullanılan formatta bir XML dosyası oluşturun.