MIT Açık Ders malzemeleri
http://ocw.mit.edu

6.046J Algoritmalara Giriş, Güz 2005

Bu materyallerden alıntı yapmak veya kullanım şartları hakkında bilgi almak için:

Erik Demaine ve Charles Leiserson, *6.046J IntroductiontoAlgorithms, Fall 2005*. (Massachusetts Institute of Technology: MIT OpenCourseWare). http://ocw.mit.edu (Giriş Tarihi: 08, 01, 2010).

Lisans: Creative CommonsAttribution-Noncommercial-ShareAlike.

Not: Lütfen atıf taparken bilgilere eriştiğiniz gerçek tarihi kullanınız.

Atıflar ve Kullanım Koşulları konusunda daha fazla bilgi için:

http://ocw.mit.edu/terms ve http://www.acikders.org.tr sitesini ziyaret ediniz.

Ders 10

Günaydın. Öyle görünüyor ki sabah 9:30 herkes için daha erken olmaya başladı. Evden bizi izleyen herkese merhaba.. Bence şöyle bir kural olmalı: Eğer bizi videodan izlemek isterseniz bunu sadece dokuz otuz onbir arası ve Pazar günleri yapabilirsiniz demeliler; veya o saatte başlamalısınız demeliler ki bizim sabahlarımızın ne olduğunu hissedebilin. Bugün dengeli arama ağaçları hakkında konuşacağız. Bu konuya bir süredir atıf yapıyorum. Bugünkü hedefimiz içinde yerleştirme, silme yada arama yapabileceğimiz ve bunları log n süresi alan işlemlerle gerçekleştirebileceğimiz bir arama ağacının veri yapısını oluşturmak olacak. Yani öyle bir ağaç istiyoruz ki bunun yüksekliği mutlaka log n olsun. Bu dengeli bir arama ağacının veri yapısıdır.

İstediğimiz veri yapısı öyle olsun ki **n** sayıdaki elemandan oluşan bir dinamik seti log n zamanında işleyebilsin. Yani söylediğimiz gibi bu ağacın yüksekliği log n düzeyinde olsun. Şimdi daha dikkatli bakarsanız aslında bir arama ağacı veri yapısını tanımlamadık; bir ikili arama ağacının veri yapısını tanımladık ve bu o türlerden sadece biridir. Bugün bunun üzerine odaklanacağız. Ve gelecek etütte de biz, daha doğrusu siz ikili olmayan dengeli arama ağaçlarına bakacaksınız.

Her düğümün ya da dalın sabit sayıda ardılı yani çocuğu olabilir; mutlaka iki olması şart değildir. Şimdi genelleme yaparak tanımlama yapıyorum. Genel anlamda bir veri ağacını daha sonra göreceksiniz. Bugün sadece ikili veri ağaçları üzerine odaklanacağız. Şimdilik bunu tanımlamayacağım. Aslında birçok değişik dengeli arama ağacı veri yapısı vardır. Şimdi söyleyeceklerim bunların arasında benim bildiklerim. Bunların birincisi büyük harfle AVL ağaçları. Bu 1962 de icat edilmişti ve bu hızlı veri yapılarının başlangıcını oluşturdu. Bundan sonraki üç tanesi aslında aynı zamanda geliştiler ve bu haftaki etütte bunları inceleyeceksiniz. Yani bunlar ikili olmayan ağaçlar. İkili olanların yanı sıra ikili ve üçlü veya iki, üç ve dördüncü dereceli ağaçlar vardır; ya da B genel kavram dereceli ağaçlar olur.

İki-üç ağaçları bulunan ikinci ağaçlardı ve 1970 de Hopcroft tarafından icat edildiler. Bugün inceleyeceğimiz ağaçlara kırmızı siyah ağaçlar denir. Bunlar garantili bir logaritmik yüksekliği olan ikili arama ağaçlarıdır. Bunların dışında başkaları da var. Gelecek hafta skip listleri yani atlama listelerini inceleyeceğiz; bunlar tam anlamıyla ağaç değil ama ağaç benzeri şeylerdir.. Ve bu haftaki problem setinizde inceleyeceğimiz treap'ler ama bunlarla ilgili fazla konuşmayacağım.

Ama bunları elde etmek bir bakıma oldukça kolay çünkü geçen hafta işlediğimiz konulara dayanıyorlar. Hatırlarsanız geçen hafta eğer rastgele yapılanmış bir ikili arama ağacı oluştururs ak bunun beklenen zaman açısından yüksekliğinin log n olduğunu görmüştük. Treap'ler de bunları dinamik yapmanın bir yoludur yani n

sayıda elemanın oluşturduğu statik bir set yerine, bu listelerde yerleştirme ve silme yapabilirsiniz ve bunları verimli rastgele permütasyonlar kullanarak bir ağacın içine yerleştirebilirsiniz. Yani bir bakıma bu en kolayıdır. Bu aynı zamanda en yakın zamanda bulunun arama ağacı yapılarından biridir. Bunu iki geometrici Rimon Sidell ve Aragen 1996 da icat ettiler. Yani bunlar arama ağacı veri yapılarının sadece bir kaçı. Bu sınıfda incelemeyeceğimiz tek tür AVL ağaçları olacaktır. Bunlar da çok zor değildir; eğer ilginizi çekiyorsa kendiniz okuyabilirsiniz çünkü bunlar eğlencelidir. Hatırladığım kadarıyla bunlarla ilgili kitapta da bir problem var.

Ama bugün biz kırmızı siyah ağaçlara odaklanacağız ve bu basit bir fikirdir. Belirli bir yolla logaritmik yüksekliği garanti eder ve böylece tüm işlemler log n süresinde desteklenir. Evet, bunlar **BST** yani ikili arama ağaçlarıdır; bunların her düğümünde bir parça ek bilgi vardır ve buna **renk alanı** yani color field denir. Bir renk alanı olan ağaca kırmızı siyah ağaç denebilmesi için karşılaması gereken bazı özellikler vardır. Bunlara kırmızı siyah özellikler adı verilir. Bunu yazmak biraz zaman alacak ama temelde bayağı kolaydır. Onun için önce bunları yazacağım, ondan sonra bunların ne anlama geldiğini açıklayacağım. Bunlarda dört özellik vardır.

Birincisi çok kolaydır. Her düğüm ya kırmızı ya da siyahtır; böylece kırmızı siyah ağaç adı ortaya çıkar. Yani renk alanı kırmızı veya siyah olduğunu belirten tek bittir. Kırmızı düğümleri çift daire içine alarak belirteceğim çünkü burada renkli tebeşirim yok ve siyah düğümleri de tek daire içine alacağım. Büyük bir ihtimalle sizin de renkli kalemleriniz yoktur onun için bu bizi bir dertten kurtaracak. Kırmızı çift daire; siyah ise tek daire olacak.. Ve biz bazı açılardan siyah düğümleri tercih ederiz. Kırmızı düğümler aslında bize bayağı sıkıntı yaratır ve bunu göreceğiz.

Evet, ikinci özellik kökün ve yaprakların hepsinin siyah olmasıdır. Ve burada küçük bir numara yapacağım; ikili arama ağaçlarını geçmişte yaptığımızdan biraz daha farklı işleyeceğim.

Normalde bir ağacı düşündüğünüzde aklınıza bir takım düğümler gelir. Her düğümün ya 0, ya 1, ya da 2 çocuğu vardır ve şöyle bir şeydir. Bir çocuğu olmayan her düğümün ardılları olduğunu hayal edeceğim ve oralara birer nokta koyacağım; bunlar harici düğümler olacak ve bunlara yaprak diyeceğim. Aslında normal olarak yapraklar şu elemanlar olmalıydı. Ben burada her olmayan çocuk işaretleyicisi yerine bir yaprak ekleyeceğim. Ve bunlar benim yapraklarım olacak. Bunlar aslında o düğümlerden gelen NIL pointers yani 0 işaretleyicileridir. Böylece her iç düğümün tam iki çocuğu olur ve her yaprağın sıfır çocuğu olur. Söylemek istediğim bunlar siyah ve ikinci kuralıma göre bu da siyah. Şimdi özellikler biraz daha ilginç hale geliyor. Her kırmızı düğümün atası siyah olacak.

Yani ne zaman bir kırmızı düğümüm olursa bunun atası siyah olmak zorunda; tek daire olmak zorunda. Böylece başka bir yaklaşımla bir ağaçtaki herhangi bir yolla baktığınızda hiçbir zaman ardarda gelen iki kırmızı düğüm göremeyeceksiniz. En

çok kırmızı siyah kırmızı siyah olabilir. Ardarda gelen siyah düğümler olabilir ama hiçbir zaman iki kırmızı düğüm ardarda olmayacak. Şimdi son bir kuralımız daha var. Bu kural bu yollarla ilgili birazcık daha açıklama getiriyor. Bu son kurala göre tüm basit yollarda, yani düğümlerden sonra vertex yani uç noktaların tekrarlanmadığı yollarda, x düğümünden x'in ardıl yaprağına kadar olan bütün yollarda mutlaka eşit sayıda siyah düğüm vardır.

Şimdi bir şekil çizmek istiyorum. Elimizde bir ağaç var; bu ağaçta bir x düğümü var. Bu x düğümünden aşağılarda olan tüm yapraklara kadar bütün yollara bakıyorum. Bu yolların hepsinde eşit sayıda siyah düğüm olmalı. Burada her birine dört tane siyah düğüm çizeceğim; yaprağın kendisi ve üç tane de onun üstünde olacak. Üçüncü özellikten biliyoruz ki bu düğümlerin en az yarısının kırmızı olması gerekiyor, çünkü kırmızı düğüm durumunda onun atası mutlaka siyah olmak zorundadır.

Ama ben bütün bu yollardaki siyah düğümlerin sayısının birbiriyle aynı olmasını istiyorum. Şurada siyah yükseklik için yeterli yer bırakmamışım; bu nedenle bunu şuraya yazacağım. Bu tüm yollar için aynı olmak zorunda ama özellikle bu sayımı yaparken x'in kendisini saymıyorum. Evet x siyahsa, buna siyah yükseklik diyorum, ve x' in siyah yüksekliği dört oluyor. Yani x kendisi de siyah olsa, siyah yüksekliği yine de dört olacak. Burada bunlar algoritmaları daha açıklayıcı hale getirmek için küçük detaylardı; şimdi bir kırmızı siyah ağaç örneğine bakalım.

Size bir örnek göstereceğim ve bu özelliklerle neden ilgilendiğimizi söyleyeceğim. Evet, bu ağacın bir takım özellikleri var. Her şeyden önce bu bir ikili arama ağacı. Bunu sırasıyla geçiş ile kontrol edebilirsiniz. Şu sayıları sıralı bir düzende size verecektir: 3, 7, 8, 10, 11, 18, 22 ve 26. Bu bir geçerli ikili arama ağacıdır. Şimdilik bu yapraklara herhangi bir anahtar koymadan ekledik. Bunlar şimdilik boşta duruyorlar. Şunlar sıfır işaretleyicileridir. Yani şunların hepsine sıfır diyebilirsiniz. Herhangi bir ardıl olmadığından bunları orada sadece işaretledik. Sonra da bazı düğümleri çift daireyle işaretledim çünkü bunlar kırmızıya boyanacak. Eğer bunu yapmasaydım o zaman siyah yükseklikler birbirlerine uyum göstermezdi. Bu nedenle biraz dikkatli olmak zorundayım. Her düğümden aşağıdaki ardıl bir yaprağa kadar olan siyah düğümlerin sayısını ölçmek istiyoruz. Örneğin 0 işaretleyicilerin siyah yüksekliği 0'dır.

İyi, bu her zaman bir cevaptır. Yani bu elemanların siyah yükseklikleri her zaman 0'dır. Bunu burada sadece belirteceğim. Bunların siyah yükseklikleri 0'dır. Pekiyi, 3'ün siyah yüksekliği nedir? 0 mı? Tam anlamı ile değil çünkü bu düğümler siyahtır. Böylece siyah yüksekliği 1 olur. Burada (3) ü saymamakla haklısınız; aslında o siyahtı ama sayılmaması gerekir. Sayımda bu hesaba katılmaz, ama yapraklar katılır. Burada sadece iki yol var ve her ikisinde de aynı sayıda siyah düğüm var ve zaten öyle de olması gerekir. Buraya bakacak olursak, örneğin 8'e, bunun da siyah yüksekliği 1, ama kendisi kırmızı. Tamam aynı şey 11 ve aynı şey 26 için geçerli. Bunların her birinin sadece iki yolu var. Ve her yolda bir siyah düğüm var.

10; bunun siyah yüksekliği nedir? Hala 1' dir, güzel çünkü biz 10'un kendisini saymayız. Burada yapraklara dört tane yol var her birinin içinde sadece 1 siyah düğüm var artı kök var ve bunu saymıyoruz. 22; aynı şeyin olmasını umuyoruz. Bu giderek daha ilginç hale geliyor. Burada bir yol var ve onun üzerinde de bir siyah düğüm var. Burada daha uzun olan başka yollar da var. Ama bunların üzerinde de sadece 1 siyah düğüm var. Yani eğer kırmızı düğümleri görmezden gelecek olursanız bütün yolların uzunluğu birbiriyle aynı. Tamam, şimdi 18, bunun daha büyük olacağını umuyorum; burada siyah yükseklik 2 olmalı, çünkü bu yolların her birinde bir siyah düğüm var ve yapraklar da da bir siyah düğüm var; veya bir siyah düğüm burada ve bir siyah düğüm de yapraklarda.

Ve son olarak da kökün siyah yüksekliğinin 2 olması gerekiyor. Burada bunu görmek daha kolay tahmin ediyorum. Bu yolların her biri 2 siyah düğüm içeriyor. Burada da aynı şey oluyor. Evet, umarım bu özelliklerin bir anlamı olur. Bunların hepsini kontrol etmedik. Her kırmızı düğümün bir siyah atası vardı. Bu yollara şöyle bir bakarsak, bir kırmızı bir siyah olarak düzenlendiğini görürüz; ya da sadece siyahlardan oluşan bir demet var. Ama hiçbir yolda iki kırmızıyı arka arkaya koymuyoruz. Kök ve yapraklar siyah, çünkü biz bunu tanım evresinde kabul ettik. Her düğüm, ya siyah ya da kırmızıdır; tamam bu kolaydır. Bu belirli bir özellikler seti ama bu evrede biraz rastgele gibi görünüyor. Bunun sonuçlarını gördüğümüz zaman daha fazla anlam kazanacak.

Ama burada gerçekleştirmeye çalıştığımız bazı amaçlarımız var. Bunlardan biri, bu özelliklerin ağacın logaritmik bir yüksekliği olmasını sağlaması ve bunun da log n düzeyinde olmasıdır. Ve bu da sağlanıyor fakat şu anda çok açık değil. Bu tüm özellikler bir araya getirildiğinde ortaya çıkacak. 3 ve 4'ncü özellikler ana özelliklerdir. Ama temelde bunların hepsine ihtiyacınız olur. Bu özelliklerden başka bir beklentimiz de bunların bakımının bir şekilde kolay olması. Aslında işin başında bu özelliği karşılayan bir ağaç yaratabilirim; örneğin biraz dikkatli olmak kaydıyla, mükemmel dengeli bir ikili ağacı alıp bütün düğümlerini siyah yaparsam, bu özelliklerin hepsini karşılayacaktır. Ama bu kırmızı siyah bir ağaçtır. İşin başında bu özellikleri sağlamak o kadar zor değildir; şaşırtıcı olan bunların bakımı konusudur.

Bu ağaca bir düğüm eklemek ve bu ağaçtan bir düğüm çıkarmak istediğimde bunun çok zor olmamasını istiyorum. log n süresinde tüm bu özellikleri yeniden yapılanmasını istiyorum. Evet, bu aslında en zor kısım olacak. İlk yapacağımız şey, bu özelliklerin ağacın yüksekliğinin mutlaka log n düzeyinde olması gerektiğini ima ettiğini kanıtlamak olacak. Böylece veri yapısındaki aramalar ve soruşturmalar hızlı çalışacaktır. Burada zor olan şey bu özelliklerin ilk baştaki özellikler değiştikten sonra da geçerli ve doğru olarak kalmasıdır.

Haydi kırmızı siyah bir ağacın yüksekliğine bir göz atalım. Böylece bu özelliklerin nereden geldiğini ve bunları neden seçtiğimizi anlamaya başlayacağız. Buradaki

iddiam, n anahtarı olan kırmızı siyah bir ağacın yüksekliğinin, (buradaki düğümleri hesaba katmıyorum çünkü sadece iç düğümleri saymak istiyorum; buradaki ekstra yapraklar ve eklediğim şeyleri de hesaba katmayacağım), en çok (2 kere n log(n+1), yani (logaritma n) düzeyinde olacağıdır. Ama burada iki düzeyiyle ifade edilen kesin bir sınırlama var. Bunun kanıtı kitapta tümevarım yoluyla verilmiştir ve bunu okumanız gerekir.

Ben burada daha çok bu kanıtın ana hatlarını vereceğim. Ama siz kesin kanıtı da, tümevarım yoluyla yapılan kanıtı da okumalısınız, çünkü tümevarım yöntemi ile kanıtlamada pratik yapmak iyidir. Buna karşılık kanıtlama skeçi sizde daha çok sezgisel bir yaklaşım geliştirir ve kırmızı siyah ağaçlarda ne olduğunu daha iyi anlarsınız ve gelecek etüde de bir giriş yaparsınız; bu şimdiden söyleyeyim. Bu tahtayı boş bırakacağım ve şuraya geçeceğim.

Şimdi yapacağım ilk iş bu ağaçta bazı değişiklikler yaparak bizim tanıdığımız bir hale getirmek. İlk yapacağım ana şey de her kırmızı düğümü, onun atasıyla birleştirmek olacak. Biliyoruz ki kırmızı bir düğümün atası siyah olmak zorundadır. Böylece her kırmızı düğümü, onun siyah atasıyla birleştireceğim. Şimdi şuradaki hale bir bakalım. Şu kırmızı düğümü alacağım onun atasıyla birleştireceğim, bu kırmızı düğümü alıp, onun yoluyla birleştireceğim ve böyle devam edeceğim. Şu yukarıda ulaşamadığım bir tane var. Onun için buraya bu şekli bir daha çizeceğim. Böylece 7 yani en üstteki düğüm şimdi 7/18 olacak. Bu ikisi birbiri ile birleşti fakat bunlardan başka birleşen olmadı.

Sonra solda 3 var. Ona katılan olmadı ve normal olarak birkaç yaprağı var. Şimdi bakarsanız, -- ama bunu bir kez daha çizmeliyim. Şimdi bu düğümleri birleştiriyorum ve buradaki düğümlerin de hepsini birleştiriyorum; çünkü bu kırmızı düğümlerin her biri şu siyah düğüme katılacak. Ve bu iki düğümü de bir araya getiriyorum. Şu kırmızı düğümü şu siyah düğümün içine koyuyorum.

Şimdi kök tarafından şöyle bir bakarsanız, bu 7/18 oldu. Bundan üç çocuk yada ardıl sarkıyor. Bunu şu şekilde çizmek istiyorum ama önce tahtayı aşağıya çekebilmem lazım. İyi, böylece 7 ile 18 arasında bu küme düğümü var; yani 8/10 ve 11'den oluşan bir birleşik düğüm var. Ve bu düğümden de sarkan dört yaprak var. Ve sağ tarafta 18 den sonra 22/26 küme düğümüm var ve bundan da sarkan 3 yaprak var. Evet, oldukça garip bir ağaç oldu çünkü bugüne kadar hep ikili ağaçlarla uğraştık ama gelecek derse bir ön hazırlık olarak bunu kabul edin. Bunun adı 2-3-4 ağacıdır. Buna neden 2-3-4 ağacı denildiği konusunda tahmini olan var mı? Evet, yaprakların dışında her düğümün 2, 3 veya 4 çocuğu ya da ardılı olabilir. Sadece yaprakların çocuk sayısı 0'dır. 2-3-4 ağaçların daha önce ima edilen hoş bir özelliği daha vardır. Aslında iki çocuğunun mu, üç çocuğun mu, dört çocuğun mu olacağının gerçek bir kontrolü yoktur; ama burada başka bir hoş özellik vardır. Evet? Bütün yapraklar aynı derinliktedir, tamamıyla doğru. Ağaçtaki bütün bu elemanların derinlikleri aynıdır; bu neden böyle olur? Dördüncü özellik nedeniyle..

Gelecek derste bu özelliğin nasıl muhafaza edileceğini göreceksiniz. Ama bu transformasyon yada dönüşümden sonra bütün yaprakların aynı derinlikte olduğu sonucu ortaya çıkar; çünkü bunların derinlikleri veya diyelim ki ağaç içindeki yükseklikleri bunların siyah yüksekliğidir. Ve bu yaprakların derinliği kökün de siyah yüksekliği olacaktır. Kırmızı düğümlerin hepsini kaldırıyoruz ve dedik ki bir yola baktığımızda ve kırmızı düğümleri ihmal ettiğimizde, o yol üzerindeki siyah düğümlerin sayısı aynıdır. Şimdi temelde bütün siyah düğümleri yerinde bırakıyoruz.

Ve böylece bütün bu yollardaki siyah düğümlerin sayısı aynı olacaktır. Bu nedenle de her yaprak aynı derinlikte olacaktır. Bu özelliklerin bir kısmını yazayım:
Her iç düğümün 2 ile 4 arasında ardılı olur ve her yaprak aynı derinliğe sahip olur, ve bu kökün siyah yüksekliğine eşittir. Bu 4'üncü özellikten kaynaklanır. Tamam, şimdi bu bize bir sürü şey anlatıyor. Aslında bu dönüşüm, temelde kırmızı düğümleri görmezden geliyor. Böylece sadece siyah düğümlere odaklandığınızda yükseklik, siyah yükseklik oluyor. Siyah yükseklik de bize kökle yapraklar arasındaki tüm yolların aynı uzunlukta olduğunu söylüyor. Bu nedenle tüm bu düğümler aynı düzeydedir. Yaprakların aynı düzeyde olması iyi bir şeydir, çünkü bu ağacınızın oldukça dengeli olduğu anlamına gelir. Eğer bir ağacınız varsa ve bütün düğümlerden dallanmalar oluyorsa ve bunların da en azından iki çocuğu varsa ve tüm yapraklar da aynı düzeyde ise o zaman bu ağaç oldukça dengeli bir durumdur. Tamam, şimdi bunun bir formunu kanıtlayacağız. Bu ağacın yüksekliğine h' diyeceğim. Orijinal ağacın yüksekliği h idi. Burada sınırlamak istediğimiz de buydu. Bu nedenle yapılacak ilk şey h' nü sınırlamaktır. Sonra da h ile h' arasında bir ilişki kuracağız.

Tamam, ilk soru bu ağaçta kaç tane yaprağım olacağı olduğudur? Ve hangi ağaca baktığım da farketmez çünkü yaprak sayısıyla ilgili herhangi bir değişiklik yapmadım. Bütün yapraklar siyahtır. Bu nedenle yapraklar hiç değişmedi. Şu ağaçta, dolayısıyla bu ağaçta kaç tane yaprak vardır? Afedersiniz? 9, evet doğru 9 tane var ama ben daha genel sordum, kusura bakmayın. Bu örnekte 9 tane var. Kaç tane anahtar var? 8, bu durumda genelde 9'u 8'in bir fonksiyonu olarak nasıl yazarsınız? 9'un veya 8'in büyük değerleri için.. Afedersiniz? Artı 1, güzel, doğru cevap tahminle geldi n+1. Pekiyi, neden n+1?

Şuradaki ikili ağaç durumuna bakarak neler olup bittiğini anlamaya çalışalım. Ne zaman bir anahtarınız varsa orada iki tane dal oluyor. Ve bu çok iyi bir kanıt değil. Burada dallanan ikili ağaç dediğimiz bir yapı var. Her iç düğümün en az iki çocuğu yani ardılı var. Ve biz yaprak sayısını, işleme alınan iç düğümlerin sayısı cinsinden hesaplıyoruz. Bir ağaçtaki yaprakların sayısı, yani dallanan bir ağaçtaki yaprakların sayısı her zaman iç düğümlerin sayısı artı 1.

Bunu bilmeniz gerekir. Bunu tümevarımla kanıtlayabilirsiniz. Evet, yaprakların sayısı n+1'dir Eğer tek çocuğunuz varsa bu geçerli olmaz. Bunun geçerli olması için her iç düğümün dallanma faktörünün 2 olması gerekir. Evet bu bir either

ağacıdır yani ne de ağacıdır ve şimdi de yaprakların sayısı ile ağacın yüksekliği arasında bir ilişki çıkarmak istiyoruz. Evet, burada kullanabileceğimiz iyi bir ilişki ne olabilir? Kaç yaprak olduğunu kesinlikle biliyoruz ve bu bir şekilde bizi n ile ilişkilendirecektir. Bizi ilgilendiren yükseklikdir ve bu ağacın yüksekliğine bakalım. Şimdi elimde bir 2-3-4 ağacı varsa ve bunun yüksekliği h' ise, bunun kaç yaprağı olabilir? Buradaki en çok ve en az yaprak sayısı ne olacaktır? 2 üzeri h den 4 üzeri h kuvvetine. yada h' kuvvetine kadar..

Böylece 2-3-4 ağacındaki yaprakların sayısının en fazla 4' ün h' kuvvetine kadar gideceğini biliriz; çünkü bir düğümden en fazla 4 dal ayrılabilir. Ve ayrıca yaprakların sayısının en az 2'nin h' kuvvetine kadar olacağını biliyoruz; çünkü her düğümde en az 2 dallanma olabilir. İşte anahtar budur. Bunlardan sadece biri ile ilgileniyorum, o da şu. Yani 2'nin h' kuvveti en çok n+1'e eşit olabilir. Yani buradaki yaprakların sayısı n+1'dir. Bunu kesinlikle biliyoruz. Bu nedenle bunu yeniden yazarız ve her iki tarafta da logaritma alırız. Bu durumda h' en çok n+1'in logaritması olabilir. Böylece hoş, dengeli bir ağaç elde ederiz. Bu yeterince sezgisel olmalı. Eğer her düğümden iki dal çıksaydı ve bütün yapraklar da aynı düzeyde olsaydı bu mükemmel bir ağaç olurdu. Bu durumda da bunun yüksekliği logaritma 2 tabanında n+1 olurdu; tam anlamı ile n olmazdı.

Bu ağacın yüksekliği olmalıdır. Burada daha fazla dallanma olabilirdi ve bu şekli bir bakıma daha da sığlaştırabilirdi. Yani aynı yükseklikte daha fazla yaprağım olabilirdi. Ama bu, sadece benim için iyidir. Bu yüksekliği yaprakların sayısı bazında azaltır. Burada n+1 yaprakların sayısı idi. Bu hoş oldu, ağacın yüksekliği ile ilgili kolay bir üst sınır elde edildi. Aslında bizim ilgi alanımız şu ağacın yüksekliğidir; bu nedenle h'yü h' ile ilişkilendirmek istiyoruz.

Bunu nasıl yapabileceğimizle ilgili herhangi bir öneri var mı? Bu indirgenmiş ağacın yüksekliğinin şundan çok da fazla küçük olmayacağını nasıl biliriz? Bunun en fazla log n olduğunu biliyoruz. Bunun da en çok 2 log n+1 olmasını istiyoruz; yani yanıtı biliyoruz. Teoremi söylemiştik. Afedersiniz? Doğru üçüncü özellik bize her siyah düğüm için sadece bir kırmızı düğümümüz olabileceğini söylüyordu. Bu durumda en fazla kırmızıyla siyahı birbiriyle değiştirebiliriz.

Eğer kökten bir yaprağa kadar giden bu yollara bakarsak, burada olabilecek kırmızı düğümlerin sayısı en çok o yolun yarı uzunluğu kadar olacaktır. Ve biz tüm yolların en büyük değerini alırsak bu ağacın yüksekliğini verecektir. Yani h'nin en çok 2 çarpı h' olduğunu biliriz. Başka bir deyişle, h üssünü en az h nin yarısı olacak şekilde düşünmek kolaylık getirir. Çünkü, bunu doğru yaptığımı varsayarsam, her kök-yaprak hattında en fazla düğümlerin yarısı kırmızı olabilir.

Yani en azından yarısı siyah olmak zorundadır. Ve bütün siyah düğümler bu şekilde gösterilmiş olduğundan, elimizde şu bağıntı olabiliyor: h en çok 2 kere log(n+1)'dir. Tamam, bu oldukça kolaydı. Ama unutmamanız gereken bir şey var; bu ağacın dengeli olması lazım ve bu ikisi birbirinden çok farklı değil. Gelecek

etütte bu formdaki ağaçlarla nasıl işlem yapacağınızı göreceksiniz. Bunu yapmanın akıllıca bir yolu var. Bunlar 2-3-4 ağaçları. Bugün biz sadece kırmızı siyah formdaki ağaçlarla işlem yapacağız. Ve bugünkü derste göreceklerinizle gelecek etütte göreceğiniz işlemler aslında ilk bakışta birbirinden ilgisiz gibi görünecek..

Ama aslında birbirleriyle aynıdırlar; sadece benzerlik biraz gizlidir. Evet, iyi haber buydu. Şimdi biliyoruz ki bütün kırmızı siyah ağaçlar dengeli. Yani, eğer ağacımızın kırmızı siyah bir ağaç olarak kaldığından eminsek, o zaman her şey yolunda demektir. Burada her şeyin yolunda olmasını ağacın yüksekliğinin her zaman log n olarak kalması şeklinde yorumluyorum. Kırmızı siyah bir ağaçta sorgulama yaptığımda ki sorgulamalar arama, belirli bir anahtarı bulma, en küçük değeri bulma, en büyük değeri bulma, bir ardılı bulma, ya da bir atayı bulmak şeklinde yorumlanabilir ve bu sorgulamaları biz ikili arama ağacında görmüştük.

Ve bunları n düzeyindeki yükseklik süresinde nasıl ele alacağımızı biliyoruz. Buradaki yükseklik de log n, onun için bir kırmızı siyah ağaçtaki işlemler de log n düzeyinde zaman alacaktır. Yani sorgulamalar oldukça kolay.

Sorgulamalarla ilgili bize avantaj sağlayan buradaki dengedir ve bu bir sürpriz değildir; biliyoruz ki denge olması iyi bir şeydir. Zor olan şey güncellemeleri yapmak olacak. Ve buradaki bağlamda güncelleme, araya sokma ve silme demektir. Genellikle veri yapılarında biz sorgulama deyince bu veri yapısındaki bazı soruları sormak ondan sonra da yapının içindeki veriyi değiştirmeyi kast ederiz. Ve çoğu zaman biz bu işi yaparken dinamik setleri düşünürüz. Dinamik setleri bir elemanı eklemek ya da çıkarmak şeklinde değiştirebilirsiniz. Burada her türlü soruyu da sorabilirsiniz. Öncelikli kuyruklarda başka türlü güncellemeler, örneğin en küçüğü sil tipi güncellemeler de olabilir.

Burada biz minimumu bulduk, sonra bunu silebiliriz. Tipik olarak bu işlemler bizi ilgilendiren işlemlerdir. Ve bunun gibi işlemleri yapmakla, güncellemeler ve sorgulamalarla ilgili konuşacağız; ilgili konu ne ise bunları konuşacağız. Problem setlerinde destekleyebileceğiniz çeşitli sorgulamalar göreceksiniz.

Tamam, biz güncellemeleri nasıl destekleyebiliriz? Elimizde ikili arama ağacı araya yerleştirmeleri vardı ve bunlara ağaç araya yerleştirmesi diyoruz.
İkili arama ağacı silmeleri var ve bunlara da ağaç silmeleri diyoruz. Bunlar ikili arama ağacının özelliklerini korur ama her zaman dengeyi koruyabildiklerini söyleyemeyiz. Birkaç tane düğümü araya yerleştirebiliriz. Yeni minimum elemanlar eklemeyi sürdürürsek, işin sonunda gerçekten bir uzun yol elde ederiz. Bu açıdan bakarsak bu uygulamalar kırmızı siyah özellikleri korumazlar çünkü biz başında kırmızı siyahın dengeyi ima ettiğini biliyoruz. Özellikle de biraz önce sildiğim birinci özelliği karşılamazlar ki bu da her düğümün ya kırmızı ya da siyah olacağıydı. Bu uygulamada bir düğüm ekleyebilirsiniz ama ona bir renk vermezsiniz. Bu nedenle ek olarak buna bir de renk atamamız gerekir. Ve bunu yaptığımız anda da büyük bir olasılıkla başka bir özelliği karşılamaz duruma

geleceğiz. Ondan sonra da bu özelliği tekrar düzeltmek zorunda kalacağız ve bu iş böyle gidecek. Bu nedenle bu biraz karmaşık ama bununla uğraştığımız zaman o kadar zor olmadığını göreceksiniz.

Yani, güncellemeler ağacın yapısını değiştirmelidir. Ve ağacın kırmızı siyah özelliklerini korumak için üç türde değişiklik yapmaya ihtiyaç vardır. İlk yapılacak şey aslında BST işlemidir yani ağaç araya yerleştirme veya ağaç silme işlemidir. Bunun nasıl yapılacağını biliyoruz. Şimdi bunu yapalım. Bunu yaparken bazı düğümlerin renklerini değiştirmek zorunda kalacağız. Özellikle de araya yerleştireceğimizin mutlaka bir rengi olması gerekir. Ve genellikle bir düğümü oradan çıkarırsak bunu ve onunla birlikte komşu düğümleri de yeniden renklendirmemiz gerekir.

Yapacağımız bir başka tür işlem daha var. Ttekrar renklendirmek bunu kırmızı ya da siyah yapmak demektir. Yapılacak diğer işlem ağacın yapısını tekrar düzenlemektir; yani buradaki işaretçileri bir düğümden diğerine olan linkleri değiştirmektir. Ve bunu da çok dikkatle yapılandırılmış bir şekilde yapacağız. Bu nedenle de kırmızı siyah ağaçlar çok ilginçtir. Bunlarda yapılan değişiklikler oldukça basittir ve çok da fazla değildir. Bunlara **rotasyonlar** denir. İşte burada bir rotasyon var.
Bu ağacın cinsine özgü yani jenerik bir kısmının çizimidir. Elimizde iki tane düğüm var, A ve B. Buradan sarkan bazı alt ağaçlar var ve bunları üçgenler şeklinde çizdik. Bunların hangi büyüklükte olduğunu bilmiyoruz. Ama bilmemiz gereken şey bunların hepsinin aynı siyah yükseklikte olduğu; böylece bu ağaç bir kırmızı siyah ağaç olur. Ama genelde böyle bir şey görünür. Burada bir ata ve ağacın içinde çizmediğimiz diğer bölümler vardır. Bu alt ağaçlara Yunanca isimler vereceğim; alfa, beta ve gamma diyeceğim. Ve ben B'nin sağa rotasyonunu yani sağa dönmesini tanımlayacağım. Genelde eğer bir B düğümüm varsa, buna bakıp bunda bir sağa rotasyon yapacaksam, bu çizimde solunda görünen ardılına ve iki düğümün alt ağaçlarına bakıp bu resmi çizerim.

Ve şu ağacı oluştururum. Yani tüm yaptığım bu kenarı doksan derece döndürmekti. Eskiden B'nin atası şimdi A'nın atası oldu. A şimdi B'nin yeni atasıdır; bu durumda alt ağaçlar tekrar organize olur. Daha önce şu ikisi A'nın alt ağaçlarıydı. Ve gamma B'nin bir alt ağacıydı. Gamma hala B'nin bir alt ağacı ve alfa da hala A'nın bir alt ağacı. Ama beta B'nin alt ağacı oldu. Burada kontrol etmemiz gereken ana husus bu işlemin ikili arama ağacı özelliğini muhafaza etmesidir. Hatırlarsanız, ikili arama ağacının özellikleri çerçevesinde, sol alt ağaçtaki elemanların hepsi bu düğümdeki değere eşit ya da ondan daha küçük ve sağ alt ağaçtaki elemanların hepsi de bu düğüm elemanının değerinden daha büyük veya ona eşit olarak belirtilmişti.

Böylece özellikle herhangi bir düğümü ele aldığımızda, örneğin alfadaki küçük a, betadaki küçük b ve gammadaki küçük c' ye baktığımızda, a büyük A'ya eşit ya da ondan küçük; küçük b'ye eşit yada ondan küçük;, büyük B'ye eşit yada ondan

küçük; küçük c'ye eşit yada ondan küçüktür. Ve bu durum hem sol tarafta hem de sağ taraftaki kuraldır; çünkü alfa her şeyin solundadır. Beta A ile B arasındadır ve gamma B'den sonra gelir.

Burada da aynı şey geçerlidir. Beta hala büyük A ve büyük B arasındaki bütün düğümleri kapsar. Ve bu iyidir. Bu işlemi kesinlikle kullanabiliriz ve sonunda hala ikili arama ağaçları bozulmaz ve burada rotasyonları çok dikkatli bir şekilde kullanarak bütün bu özelliklerin yok olmamasını sağlarız. İşin zor kısımı da budur. Ama rotasyonlar bizim anahtarlarımız olacak. Burada yaptığımız sağa dönme işlemi yada rotasyonuydu.

Bunun tersi sola dönme işlemidir. Şu büyük A'nın sola dönmesidir. Genelde işleme alınan iki düğüm arasında üsttekini listeleriz. Yani B'nin sağa rotasyonu size bunu verir. A'nın sola rotasyonu da size bunu verecektir. Bunlar tersinebilir işlemlerdir ve bu da iyi görünüyor. Bir başka açıdan bunlar sabit zaman alan işlemlerdir -- çünkü burada sabit sayıda işaretçiyi değiştiriyoruz.

B düğümünü bildiğiniz sürece bunun beta olmasını hala istiyorsanız ve bununla ilgileniyorsanız soldaki işaretçiyi B'nin soluna A'nın sağına gelecek şekilde yerleştirirsiniz ve bu işlem böyle devam eder. Bu değişikliklerden sabit sayıda yaparsınız. Bu arada ataları da güncellersiniz. Burada değiştirdiğiniz linklerin sayısı sabittir, bu nedenle yapacağınız işlem sayısı da sabittir. Belki bu rotasyonları ya da dönme işlemlerini daha önce görmüşsünüzdür. Ama biz bunları karmaşık bir yolla kullanacağız.

Şimdi araya yerleştirmeyi nasıl yapacağımıza bir bakalım. Bir bakıma bunu üç kez göreceğiz. Önce size temeldeki düşünceyi anlatayım ve bu oldukça kolay. Bunun bir kısmını daha önce belirtmiştim. Sonra bu konuda bir örnek yapacağız ki bunu iyice hazmedebilin ve ondan sonra da bunun sözde kodunu size vereceğim ve eve gittiğinizde isterseniz onu uygulayabilirsiniz. Tamam, şimdi buna kırmızı siyah araya yerleştirme diyorum; kitapta buna RB araya yerleştirmesi deniliyor; bu RB root birasının kısaltılmışı değil, kırmızı siyahın kısaltılması olarak algılanmalı.

Tamam, şimdi yapacağımız ilk şey daha önce söylediğim gibi ikili arama ağacı mantığında şu düğümü araya yerleştirmek. Bu durumda x bir yaprak olur. Biz x'in gitmesi gerektiği yer konusunda bir arama yaptık. Buna bir yaprak dememeliyim. Şu anda sarkan bir -. Bu bir iç düğüm ve orijinal düğümlerden birinden sarkıyor. Belki bunu tam şuraya ekledik. Bu durumda ondan sarkan iki tane yeni yaprak olur. İç ardılları yok. Ve bunun için bir renk seçmeliyiz. Ve bu rengi de kırmızı olarak seçeceğiz.

Pekiyi neden kırmızı seçtik? Çünkü iki renkten birini seçmek zorundayız. Burada yazı tura atabiliriz. Bu sonuç verebilir ama bizim işimizi daha karmaşık hale getirir. Evet, buraya yeni bir düğüm ekliyoruz. Bu bir kök ya da yaprak olmak zorunda değil, bu nedenle ikinci özellik nedeni ile siyah olması da şart değil.

Üçüncü özelliğe göre her kırmızı düğümün bir siyah atası olmak zorunda. Bu bir problem olabilir. Problem eğer atası kırmızıysa oluşur. O zaman ikinci afedersiniz üçüncü özelliğe aykırı düşmüş oluruz. Ama iyi haber dördüncü özelliğin hala karşılanması, çünkü dördüncü özellik sadece değişik yollardaki siyahları sayar. Bu aslında karşılanması bayağı zor bir özelliktir. Bu nedenle eğer kırmızı bir düğüm eklersek siyah yüksekliklerin hiç birisi değişmeyecektir.

Yoldaki siyah düğümlerin sayısı hiçbir şekilde değişmiyor. Ve bunun da geçerli olması lazımdır. Burada karşılayamadığımız tek özellik üçüncü özellik olabilir. Ve bu kabullenilebilir. İşin başında bir şeyleri bozacağımızı biliyorduk. Sadece bir ikili ağaç araya yerleştirmesi yapıp kurtulamazdık. Tabii şimdi bu ağaçta bunu bir deneyelim. Yani bunu düzeltmeyi deneyelim.

Üçüncü özelliği nasıl düzeltiriz? Üçüncü özellikteki ihlali ağaç boyunca yukarıya doğru taşırız. Yani x düğümünde başlar bunu yukarıda köke doğru taşırız. Bunu yeniden renklendirmeyle yaparız. Önce yeniden renklendirme yapacağız ve bir noktaya geldiğimizde ihlali rotasyon kullanarak düzelteceğiz; belki de yeniden renklendirme yapmamız gerekecek.

Şimdi bu algoritmayı uygulamada görelim. Ben bu ağacı buraya kopyalayacağım ve sizin de kopyalamanız gerekecek. Yani ilk şeklin üzerinde değişiklikler yapmak yerine bunu yeniden çizeceğim. Şimdi bu güzel kırmızı siyah ağacı oluşturduk ve araya yeni bir 15 değeri yerleştireceğiz. 22 siyah ve 22 yeni siyah olacak. Aslında bu ağacın eskisiyle aynı olması gerekiyor. Şimdi araya yerleştirmek için seçtiğim sayı 15 çünkü bu oldukça ilginç bir araya yerleştirme durumu ortaya çıkaracak.

Bazen araya yerleştirmeler çok iş gerektirmez. Rotasyonu yaparız ve işimiz biter. Ama burada ilginç bir duruma bakmak istiyorum. Bu nedenle 15'i araya yerleştiriyorum. 15; 7'den büyük. 18'den de küçük. 10'dan büyük. 11'den büyük. Bu nedenle 15 buraya gelir. Bu nedenle buraya yeni bir kırmızı düğüm, 15 düğümunu ekliyoruz. Bu durumda bundan sarkan iki tane siyah yaprak var ve bir tane siyah yaprağın yerine geçti. Bu durumda üçüncü özelliği ihlal ettik çünkü kırmızı bir düğüme yeni bir kırmızı ardıl ekledik.

Bu durumda elimizde kökten yaprağa giden hatta arka arkaya iki tane kırmızı düğüm oldu. Şimdi bunu siyah yapmak isteriz ama bu siyah yüksekliklerin düzenini tamamen bozar çünkü bu düğümün bu yolda bir siyah düğümü, ve bu yolun üzerinde aşağıya doğru iki tane siyah düğümü olur. Ve bu iyi değil. Ne yapabiliriz? Önce yeniden renklendirmeyi deneyelim. Evet bunu hatırlamak her zaman biraz zaman alır.

Evet bizim buradaki düzeltmemiz yeniden renklendirme olacak. Benim aklıma gelen ve doğru olmayan ilk şey yeniden renklendirmeyi buralarda yapmaktı. Ama bu çok iyi görünmüyor, çünkü buralarda bazı kırmızılarımız var ve şurada da bir siyah düğümümüz var. Bu nedenle bunu kırmızı şunu da siyah yapamayız. İyi

çalışmaz böyle bir şey. Şimdi daha yukarıya 15'in büyük atasına bakacak olursak burada bir siyah düğüm ve şurada da iki tane kırmızı ardıl var.

Bu aslında iyi bir haber çünkü bu durumda şunları iki siyah ardıl ve bir kırmızı ata haline getirebiliriz. Yerel olarak bu oldukça uygun olacak. Bu uygulama aslında siyah yüksekliğini değiştirmeyecek çünkü bu düğümlerden geçecek herhangi bir yol aynı sayıda siyah düğümlerden geçecektir. Yani buradaki siyah düğümden geçmek yerine ya şuradaki siyah düğümden veya buradaki siyah düğümden geçecek, çünkü yollar her zaman yapraklara doğru gider.

Evet yapacağımız şey bu bunların rengini değiştirmek olacak. Ve bu durumda kırmızı olan 10'u elde edeceğiz, sonra 8 siyah ve 11 siyah olacak ve bunlar değişmeyecek. Diğerlerinin hiç birisi değişmeyecek. Bu durumda 15'i kırmızı bırakacağız; artık ihlal ortadan kalktı. 15 uygun oldu, çünkü bu durumda atası siyah oldu. Şimdi, yeni bir ihlalimiz oluştu; burada 18 var ve 18 kırmızı. Ve elimizde kalan tek ihlal de bu. Genelde bunu yaparken sadece bir ihlal olur ve onu düzeltene kadar da bu devam eder. Sonunda da sıfır ihlal elde ederiz. Şimdi 10 ile 18 arasında bir ihlal var. Bu bana her zaman sezgiyle anlaşılması zor gelmiştir.

Şimdi şu kopya kâğıdıma bir daha bakayım. Gerçekten mi? Hayır değil. Diyecektim ki biz artık yeniden renklendirme yapamayız. İyi o kadar da kötü değilim. Şimdi yapacağımız şey tekrar 10'un büyük atasına bakmaktır ki bu da 7 ve ağacın köküdür. Bu siyah ama ardıllarından biri siyah diğeri kırmızı. Bu nedenle aynı oyunu burada oynayıp 7'nin siyahlığını kaldırıp bunu altındaki iki ardıla aktaramayız.

Kökün siyah olarak kalması konusunu şimdilik unutun. Bu özelliği şimdilik görmezden geleceğiz. Bu ikisini siyah yapıp şunu kırmızıya çeviremiyoruz çünkü bu durumda bir dengesizlik ortaya çıkıyor. Bu zaten siyah. Böyle bir durumda buradan aşağıya giden yollarda şu yolu takip eden duruma göre bir eksik siyah düğüm olur. Yani 7'yi veya onun ardıllarını yeniden renklendiremeyiz. Bu durumda bir rotasyon yapmamız gerekir. Ve bunu da sona doğru yapmamızda yarar var. Bu nedenle rotasyonu şu kenarda yapacağım. 18'i sağa doğru döndüreceğim.

Bundan sonraki işlem ne olur? 18'in sağa dönmesi. Bundan sonra bir işlem daha yapacağız. Önce 18'i sağa çevirelim. Kök olduğu gibi kalır; yani 7, 3 ve onun çocukları. 7'nin sağ ardılı artık 18 değil. Şimdi bu değer 10 oldu 18 de 10'un kırmızı ardılı oldu. Evet, burada 8 ve onun iki ardılı var. 11 ve 15; bu alt ağaç 10 ile 18 arasına sığar. Şöyle olur: 11 ve 15. Sonra da sağdaki alt ağaç var. 18'in sağında olan her şey şuraya girer: 22 ve 26. Ve umuyorum ki bu işlemler sırasında herhangi bir renk değiştirmiyorum. Eğer yaptıysam lütfen beni uyarın.

Evet, iyi görünüyor. Ama hala bu ihlal ve 10 ile 18 arasında sorun var. Ama bunu daha düzgün hale getirdim. Evet yapmak istediğimiz şey buydu; 18 ile ihlali

yaratan onun büyük atası arasındaki bağlantıyı doğrusal bir bağlantı haline getirmek: iki sağ ve iki sol. Burada solda bir sağ sol zikzağı vardı ve bunu düz hale getirmek istedik. Sonunda baktığımızda şundan daha dengeli bir ağaç gibi görünmüyor. Aslında biraz daha kötü gibi görünüyor.

Şimdi yapabileceğimiz şey şunları döndürmek, ya da daha doğrusu şu kenarı döndürmek olabilir. Bu durumda 7'yi sola döndüreceğim 10'u kök yapacağım ve ondan sonra durum daha dengeli görünecek. Bu 7'nin sola rotasyonudur. Bu evrede bir de yeniden renklendirme yapacağım, çünkü bir şekil daha çizmek istemiyorum ve kök siyah olmak zorundaydı. Bu nedenle 10'u hemen siyaha çevireceğim. Ve 7'yi kırmızı yapacağım. İşte değişiklik bu. Ve geri kalanı sadece rotasyon olacak. Şimdi 18 şurada olacak. Buna göre burada da bir rotasyon yapmam lazım çünkü kırmızı siyahlık olgusuna dikkat etmem gerekiyor.

8, 7 ile 10 arasındadır; onun için buraya gider. 11, 10 ile 18 arasına gireceğinden şuraya gider. 22 ve 26 18'den sonra gelir. Ve eğer şanslıysam istediğim bütün özellikleri karşılamış olurum. Her düğüm ya kırmızı ya siyah. Her siyah düğümün bir ardılı var. Bu en son değişim yaptığımız yer. Kırmızı düğümlerin siyah ardılları olması gerekiyor ve bütün siyah yüksekliklerin iyi tanımlanmış olması gerekiyor. Her düğümden yaprağa kadar olan yolda siyah düğümlerin sayısının aynı olması gerekiyor. Ve bunu kontrol ettiğinizde,ki daha önceden doğruydu, burada yeniden renklendirmeyle biraz kurnazlık yaptım ama hala doğrudur.

Yani bunu sadece bu rotasyonun etrafında yerel olarak da kontrol edebilirsiniz. Bunu birazdan yapacağız. Şimdilik şu örnek üzerinde çalışalım. Bu yeniden renklendirme ve rotasyonların nereden geldiği o kadar açık olmayabilir ama gördüğünüz gibi çalıştı ve en azından sizi bunun mümkün olabileceği konusunda ikna etti. Şimdi bunu yapmak için genel bir algoritma vereceğiz. Devam etmeden önce sorular var mı? Yani demek istiyorum ki algoritmayı sadece yazmak yeterince sezgisel olmuyor.

Kırmızı siyah ağaçlar sizin biraz oynayabileceğiniz şeylerdir. Dersiniz ki sadece yeniden renklendirme ve rotasyonlarla ilgileneceğim. Kendimi bu işlemlerle sınırlayacağım. Ne yapabilirim? Önce yeniden renklendirmeyi denerim eğer bu çalışırsa o zaman problemi daha yüksek düzeye taşır. Ve burada sadece log n düzeyleri vardır; yani sonunda bu işlem log n düzeyinde zaman alır. Belli bir noktada tıkanırım. Artık yeniden renklendirme yapamam. Bu durumda da birkaç tane rotasyon işi görecektir.

Her zaman iki rotasyon yeterli olacaktır. Siz bunlarla birazcık oynarsanız sonunda işlediğini görürsünüz Ve nasılını burada açıklayacağım. Varsayalım ki elimizde bir kırmızı siyah ağaç var ve x değerini araya yerleştirmek istiyoruz. Algoritması burada. Önce bunu BST yani ikili arama ağacında araya yerleştireceğiz; bildiğimiz bir şey olduğu için. Sonra bu düğümü kırmızıya boyayacağız. Şimdi burada biraz daha kesin bir simgelem kullanacağım. Renk x'in bir alanı olacak. Ve sonra bir "**while**" döngüsüyle ağaç içerisinde yukarıya

doğru köke ya da siyah bir düğüme gelene kadar yürüyeceğiz. Yani genelde x başta bizim araya yerleştireceğimiz eleman olacak; ama sonra ağacın içerisinde yukarıya doğru yürüyeceğiz
.
Eğer x'in bir siyah düğüm olduğunu görürsek bundan mutlu olacağız çünkü belki bunun atası kırmızıdır. Belki de değildir. Ama bu beni ilgilendirmiyor. Siyah düğümlerin rastgele renkli ataları olabilir. Bizim dert edindiğimiz kırmızı düğümler. Eğer x kırmızıysa bu döngüye devam etmeniz gerekir. Tabii bu arada yanlış bir şey yazdım. Eğer rengimiz kırmızıysa bunu yapmaya devam edeceğiz. Böylece ortada 3 durum olur; nasıl saydığınıza bağlı olarak 6 durum da olabilir. Bu nedenle bunu ezberlemek biraz karışıktır. Ama ortada bazı simetrik durumlar da var. Bunları şimdi çizeyim. Bizim ilgi alanımız daha önce söylediğim gibi x ile onun büyük atası bölgesindedir.

Burada kısa olması nedeniyle p[x]i kullanıyorum ve bu da x'in atası anlamına gelecek. Yani p[p[x]], x'in büyük atasıdır. p[p[x]]'in solundaki sol ardıldır. Burada x ile ilgilendiğim için ona bakarım ve herhangi bir yön tayin etmem. x, p[x]'in bir ardılıdır ve p[x] de daha öncekinin yani p[p[x]]'in ardılıdır. Şimdi bu kenarlar dikey değildir; ya sola ya da sağa eğimlidirler. Ve ben hangisi ile ilgileneceğim? Burada bakmam gereken atanın büyük atanın sol ardılı olup olmadığı. Yani bilmek istediğim, bunun gibi mi görüneceğidir.

Bu evrede x'in atasının solunda mı sağında mı olduğunu bilmiyorum, ama x'in atası p[px]]'in sol ardılı mıdır, yoksa sağ ardılı mıdır? Ve bu iki durum birbirine tamamen simetriktir. Ama bunu bir şekilde varsaymam gerekiyor, aksi takdirde bu resimleri çizemem. Tamam, şimdi bu durumda şuna kategori A diyelim. Ve bu da kategori B olsun . Ve size kategori A'da ne yapılacağını söyleyeceğim. Ve kategori B de simetrik olacak. Yani sol ile sağ arasında gidip gelebilirsiniz. Tamam, şimdi bu A. Kategori A'nın içinde 3 tane durum oluşur. Kategori B'nin içinde de benzer ama tersinden 3 durum oluşur.

Bu durumda büyük atanın diğer çocuklarına bakacağız. Bu hangi yöne baktığımızı neden bilmemiz gerektiğine bir açıklık getirir. Eğer x'in atası büyük atanın solunda kalıyorsa, o zaman büyük atanın öteki ardılına yani büyük atanın sağında kalan ardıla bakmamız gerekiyor ve buna y düğümü diyelim. Bunun başka bir ismi, y'nin erkek ya da dişi olmasına bağlı olarak, x'in amcası ya da halasıdır. Evet, bu amca ya da hala. Maalesef İngilizcede bunun cinsiyete bağlı olmayan benim bildiğim kadarıyla bir versiyonu yoktur. Yani çocuk ya da ebeveyn var ama amca ya da hala yok. Ama eminim ki biz bir tane buluruz. Ama burada denemeyeceğim, çünkü kulağa hoş gelmez. Pekiyi, y ile neden ilgileniyorum? Çünkü ben bu yeniden renklendirme adımını yapıp yapamayacağımı bilmek istiyorum.

Yeniden renklendirme düşüncesinde hatırlarsanız, büyük atalar diyelim ki bu siyah. Eğer büyük atadaki siyahlığı aşağıdaki 2 ardılına taşıyabilirsem, yani bunların ikisi de kırmızı ise, o zaman mutlu olacağım. Bu nedenle problemi

yukarıya taşırım. Buradaki öğe şimdilik kırmızı.. Buradaki de siyah.. Bu nedenle bu ikisi şimdilik uygun görünüyor. Şu eleman büyük büyük atayı ihlal edebilir. Ama öyle bir durumda da yukarıya çıkmaya devam ederiz ve o zaman işler düzelir. Bugün eğer şanslıysak, y kırmızı olacaktır.

Bu durumda yeniden renklendirme yapabiliriz. Eğer y'nin rengi kırmızıysa o zaman bizi bunu yeniden renklendirebiliriz. Ve ben bunu durum 1 adını verdiğim bir şemada göstereceğim. Ama öncelikle size bu durumların nasıl oluştuğunu anlatayım; sonra da onların nasıl çalıştığına bakalım.

Yani eğer durum 1 de değilsek bu "**else**" yani "diğer" durum komutu bununla uyumlu olacaktır ve bu durumda ikinci veya üçüncü durumda olacağız. İşte **dikotomi** ya da ikiye bölme işlemi buradadır. Aslında biz bütün durumları gördük; belki A'nın B'ye karşıtlığını görmedik ama en başta sadece yeniden renklendirme durumunu görmüştük. Bu Durum 1'dir.

Gördüğümüz ikinci şey büyük ata ve 10, yani 7 ve 10'un bir düz çizgide olamamasıydı ve bu bizi rahatsız etti. Zikzaklı bir biçimde yerleştirilmişlerdi. Böylece ikinci durum bunların zikzaklı durumunu kapsar. Eğer x atasının sağ ardılıysa ve ata da büyük atanın sol ardılıysa, bunun zikzak durumunu meydana getirdiğini varsaydık. Bu ikinci durumdur.

Evet son durum da x'in atasının sol ardılı olmasıdır. Bu durumda x, x'in atası ve x'in büyük atası arasında bir sol zincir elde edilir. Bu da üçüncü durumdur. Buraya "else" yazmadım çünkü ikinci durumun yaptığı şey 3. Duruma indirgenmektir. Yani 2. Durumda buradaki işlemleri yapacağız sonra şuradaki işlemleri yapacağız. 3. Durum için ise sadece şuradaki işlemleri yapacağız. Birinci durumda ise buradaki işlemleri yapacağız. Böylece bu A tarafındaki üç durumu bitirecek ve şuradaki "eğerden" "else"e geçerek devam edeceğiz. Burada söylenecek diğer şey bunun B durumu olduğu ve A durumu ile aynı ama sol ve sağ kavramları açısından A'nın tersi olduğudur. Bu doğal olarak oluşur.

Burada her sola bir şey yazdığımızda şurada onun sağına yazacağız veya tersini yapacağız. Bu aslında sadece işi tersyüz etmek gibi bir şey. Şu anda kategori A'ya odaklanacağız. Ve 3. durumda neler yaptığımızı göreceğiz. Bunu bir örnekte görmüştük. Ama şimdi jenerik olarak bunu yapalım. Bunu burada yapalım. Afedersiniz algoritmaya eklenmesi gereken bir satır daha olduğunu söylemeliyim. Çünkü burayla uyumlu değil. Kökü renklendiriyoruz. Burada olan şeyleri yaptıktan sonra kökün kırmızıya dönüşme olasılığı var. Ama biz kökün her zaman siyah olmasını istiyoruz.

Kök eğer kırmızı olursa algoritmanın en sonunda bunu siyaha dönüştüreceğiz. Bu siyah yükseklik özelliğini değiştirmez. Bu durumda her şey uygun olur çünkü her yol ya köke gider veya gitmez; yani x - yaprak yolu. Bu nedenle kökü kırmızıdan siyaha çevirmek bir sorun oluşturmaz. Bu her şeyin siyah yüksekliğini arttıracaktır ama bütün yollar hala aynı değerde olacaktır; yükseklik bir artacaktır.

Evet şimdi üç duruma bakalım. Ve burada birkaç simgelem kullanmak istiyorum. Hatırlarsanız rotasyonu tanımlamamız sırasında rastgele alt ağaçları belirtmek için üçgenler kullanmıştık. Ben bu alt ağacın siyah bir kökü olduğunu anlatmak için üçgenin üzerine bir nokta koyacağım. Burada kara tahtadayım onun için beyaz bir şey koyarsam bu aslında siyah anlamına gelecek kusura bakmayın. Ayrıca bu üçgenlerin aynı siyah yüksekliğine sahip olması gibi bir özelliği de var.

Böylece siyah yükseklik özelliğinin, yani 4. özelliğin korunduğuna dair emin olmam sağlanacak. Şimdi size 1. Durumu göstereyim. Bu işlemi yaparken 4.özelliğin her zaman korunmasını isteriz, çünkü bunu tekrar eski hale getirmek çok zordur; Bu ağacın dengesini ifade eder. Şimdi C adlı bir düğümümüz olsun, sol ardılı A, sağ ardılı B olsun ve bunlardan da birkaç tane alt ağaç sarksın. Ve bütün bu alt ağaçların siyah yüksekliği aynı olsun. Başka bir ifadeyle bunların hepsi aynı düzeyde.

Bu tam anlamıyla benim istediğim şey değildi, bağışlayın. Benim düşündüğüm bu düğümün_x olmasıydı. x kırmızı ve onun atası da kırmızı. Bu nedenle bir şeyi düzeltmemiz gerekiyor. Buradaki y düğümüne burada bakarız. Ve ben buna D anahtarını ayatacağım. Ama düğümün adı y. Ondan sarkan alt ağaçlar da var ve hepsi de aynı siyah yükseklikte. Ancak bu şekilde doğru olabilir. Bütün bu düğümler kırmızı olsaydı bütün şu düğümlerin siyah yüksekliği aynı olurdu. Bu nedenle bütün ardıl alt ağaçların, ki bunların hepsinin siyah kökleri var, siyah yüksekliklerinin de aynı olması gerekir. Evet, bu durumda büyük bir kırmızı ardıl alt ağaç grubuna bakıyoruz; bu bir siyah düğümden sarkıyor ve içindeki bir çok öğe de kırmızı.

Birinci durumda kırmızı neden bu kadar yoğun? Bunun nedenini düşünürsek, 2,3,4 tipi ağaç yaratmaya çalıştığımızda bütün bu şeyleri tek düğümün içine sığdırırdık. Burada temelde yaptığımız şey buydu. Bu 2,3,4 tipi bir ağaç değil ama, şimdi 5 ardılımız var ve bu da kötü. Bu nedenle bunu düzeltmeye çalışıyoruz. Böylece birinci durumda yeniden renklendirme yapacağız. Önce C'yi alacağız. C'yi siyah yapmak ve A ile D'yi kırmızı yapmak yerine A ile D'yi siyah ve C'yi kırmızı yapacağız.

Böylece C kırmızı olur. A siyahtır. D de siyahtır. Ve alt ağaçlar aynıdır. B aynıdır. Hala kırmızıdır. Şimdi 4. özelliği koruyup korumadığımızı kontrol etmek lazım; yani bütün yollarda aynı sayıda siyah düğümün olduğuna bakmamız lazım. Bunun olması gerekiyor çünkü alt ağaçlara dokunmadık. Onların hepsinin aynı siyah yüksekliği var. Bu durumda eğer herhangi bir yola bakarsanız, örneğin A'da olan tüm yollara bakarsanız bunların eşit siyah yükseklikleri olması gerekiyor. C'den çıkan tüm yolların da siyah yüksekliklerinin bu değer artı 1 olması gerekiyor, çünkü tüm sol yollarda bir siyah düğüm var ve tüm sağ yolarda da bir siyah düğüm var. Bu nedenle tüm siyah linkler birbirinin aynı. Böylece 4. özellik korunmuş olur. Ve bu da 3. özelliği yerel olarak düzeltti, çünkü B eskiden A'yı ihlal ediyordu.

Şimdi B hiçbir şeyi ihlal etmiyor. Ancak şimdi C ihlal ediliyor olabilir. Yapmamız gereken şey, yeni değerimiz x'i C'ye atamak. Daha önce B'deydi. Bunu birkaç düzey yukarıya taşırız. Aynı şekilde orijinal ağaçta da birkaç seviye yukarı gideriz. Böylece ağaç boyunca yavaş yavaş ilerleriz. Ondan sonra bu döngüyü sürdürürüz. Bu durum 1'dir: yeniden renklendir ve yukarıya git. C kendi atasını ihlal edebilir ve bu durumda özyineleme yapmamız gerekir. Yani bir şekilde özyinelemeyi C içinde südürüyoruz.

Şimdi ikinci duruma bakalım. Aslında hala bir bakıma bu algoritmayı resimlerle tanımlıyorum. Bu güzel bir grafik programlama dili. Haydi, 2. durumu çizelim. Ama bu noktada 1. durumla ilgili bir şeyi söylemeyi unuttum. Buralara bir şeyler çizdim. Doğru olduğunu bildiğim şey nedir? Şimdi terse çevirdiğim algoritmaya şöyle bir bakalım. Unutmayın, henüz kategori A'da olduğumuzu varsayıyoruz. Başka bir deyişle ata büyük atanın sol ardılı. Yani A, C'nin sol ardılıdır. Bunun böyle olduğunu zaten biliyordum. Bu nedenle y sağ ardıl oluyor. D ise C'nin sağ ardılı.

B'nin sağ ardıl mı yoksa sol ardıl mı olduğunu gerçekte bilmiyordum. Bu fark etmiyordu. 1.durumda bu fark etmez. Bu nedenle "A'nın ardılları terse çevrilebilir" demeliydim. Ama aynı çizimi kullanabilirim. Bunu anlatmamın nedeni 2. durumda bunun önemli olacağıdır. Yani 1. Durumda bununla ilgilenmiyorduk. 2.durumda diyoruz ki x atasının sağ ardılı mıdır yoksa sol ardılımıdır?

Eğer sağ ardılıysa o zaman 2. durumdayızdır. Bu durumda buradaki x'in, ki bu B olarak gösteriliyor, A'nın sağ ardılı olduğunu bilebilirim. Daha önce bilmiyordum ve ilgilenmiyordum. Şimdi bunun böyle olduğunu varsayıyorum. Tamam y hala şurada. Ve şimdi biliyoruz ki y siyahtır. Evet y burada ve siyah bir düğüm. Şimdi o sıkıştırma numarasını yaparsam bütün düğümler, yani A, B ve C tek bir düğüm halinde bir araya gelecekler. Ve bu durumda sadece 4 ardılım olacak. Bu gerçekte iyi görünür. y neden işin içinde değildir, çünkü siyahtır. Bu nedenle bu durumda biz A'da sol rotasyon ya da dönme yapacağız.

Böylece bu kenarı 90 derece çevririz. Elde edeceğimiz şey A'nın solda, B'nin de hala sağda olmasıdır. Bu düzen, sırasıyla geçiş olgusunu korur ve C hala tepededir. y alt ağacı daha önce olduğu gibi buradan sarkar. Öteki 3 alt ağaç B'den ve diğer ikisi de A'dan sarkıyor durumdadır. Yani bu sadece bir jenerik rotasyon şeklidir ve bu kenara uygulanmıştır. Bunun yaptığı iş daha önce x ile onun büyük atası arasında bir zikzak vardı; şimdi *zigzig* var. Yani x ile büyük ata arasında düz bir yol var.

x hala burada, x'i değiştirmiyorum çünkü 2. Durumu yaptıktan sonra hemen 3. Duruma geçeceğim. Bu 3. Durumun nasıl göründüğüdür ve şimdi 3. Durumu ele alacağım.

Sonunda 3. Duruma geldim. Ve bu araya yerleştirme algoritmasının sonu olacak. Elimizde bir siyah düğüm var; C düğümu. C'den çıkan bir sol kırmızı ardıl yani çocuk var. Bir kırmızı sol torun ve x var. Ve bunlardan başka şu siyah altağaçlar var; bunlar aynı siyah yükseklikte sarkıyorlar yani 2. Durumun sonunda elde ettiğimiz sonucun aynısı.

Bu tamamıyla ilişkili olarak buraya geçiyor. Ve unutmayın bu, kategori A'da kalan tek durumdu. Kategori A'da biz x'in atası olan B'nin, büyük ata olan C'nin sol ardılı olduğunu kabul etmiştik. Yani bunu biliyoruz. 1.durumu yaptık buradaki y kırmızı olmuştu. Bu 1. Durumdu. Şimdi biz y'nin siyah olduğunu varsayıyoruz. Ve x'in sol ardıl mı yoksa sağ ardıl mı olduğuna bakıyoruz. Eğer sağ ardılsa bunu sol ardıl haline dönüştürüyoruz. x gerçekte burada değişti. Önceden x B idi. Şimdi x A. Durum 3 te ise büyük atanın sol çocuğu olan atanın sol çocuğudur. Bu bizim incelememiz gereken son durum. Ve yapacağımız şey önceki örnekte yaptığımız gibi bir rotasyon yapmak.

Bu 3. Durum. C'nin bu durumunda bir sağ rotasyon yapacağız. Ve aynı zamanda yeniden renklendirme yapacağız. Böylece ne elde ederiz? Bu durumda B kök olur. Ve bu nedenle onu siyah yapacağım. Hatırlayın bu alt ağacın köküdür; buradan sarkan başka şeyler de var. Aslında bu resimlerde ek ataları da çizmem lazımdı. Bunlar ağacın ortasında bir yerlerde olmalıydı. Yerlerini şimdi bilmiyorum. Bu bir sağ dal olabilir: aynı zamanda sol dalda olabilir. Şu anda bilmiyoruz. C, B'nin ardılı olur ve onu kırmızı ardıl olarak işaretleyeceğim. A, daha önce olduğu gibi B'nin ardılı olur ve o nedenle onu kırmızı bırakacağım. Bunun dışında her şey buradan sadece sarkar durumda olur.

Böylece ortada 4 alt ağaç vardır ve bunların hepsinin siyah yüksekliği aynıdır. Özellikle şu sonuncusu y'yi içeriyordu ama biz artık y ile ilgilenmiyoruz. Şimdi artık iyi durumdayız çünkü başka ihlallerin olmaması gerekiyor. Daha önce x ile onun atası A ve B arasında ihlal vardı. Aslında A ve B'nin hala bir ata ardıl ilişkisi var ama B şimdi siyah ve B siyah olduğu için bunun atasının neye benzediği bizi ilgilendirmiyor. Kırmızı da olabilir, siyah da olabilir. Her ikisi de olabilir. Kabul edilebilir. Artık 3. Özelliği ihlal etmiyoruz. Bu durumu çözmüş olmamız gerekiyor.

3 özellik şimdi artık geçerli. Eğer isterseniz x artık bu düğüm oldu diyebilirsiniz. Döngü de size "iyi artık x kırmızı değil" der. Bu nedenle işim bitti. Aynı zamanda 4. Özelliğin de bu işlem sırasında korunduğunu kontrol etmemiz lazım. Ve yine bu o kadar zor değil çünkü 2-3-4 ağaç transformasyonu oldu. Eğer bütün kırmızıları ataların içine entegre edersem, bunun dışında her şey sabit olur yani aynı siyah uzunluğa sahip oldukları için ağaç içinde de aynı uzunlukta olurlar. Ve burada da bu hala geçerli. Burada buna biraz daha dikkat etmek gerekiyor çünkü burada aynı zamanda yeniden renklendirme yapıyoruz. Ama bu ağaçtaki herhangi bir yola bakarsanız daha önce bu yol siyah düğüm C'den geçiyordu, sonra da beni ilgilendirmeyen bazı kırmızı yerlerden geçiyordu. Sonra da bu ağaçların içinden geçiyordu ve hepsi siyah yükseklikleri nedeniyle birbirlerinin aynıydı.

Şimdi burada gelip adı B olan bir siyah düğümden geçersiniz. Ondan sonra bazı kırmızı düğümlerden geçersiniz. Bu çok fark etmez. Ama burada aşağı doğru gittiğiniz tüm ağaçların siyah yükseklikleri aynıdır; yani aynı düğümden başlamak kaydıyla bu ağaçtaki her yolun siyah yüksekliği aynıdır. Böylece biz 4. Özelliği korumuş oluruz ve 3. Özelliği düzeltiriz. Bu araya yerleştirme algoritmasıdır. Oldukça uzundu. Bu büyük bir olasılıkla sizin ezberlemeniz gereken bir şeydir.

Birkaç örnekle denerseniz o kadar zor olmadığını göreceksiniz. Bu örnekte yaptığımız her şeyin 3 durumla ilgili olduğunu görebiliriz. Birinci adımı yersizlik nedeniyle maalesef sildim ama, yaptığımız tek şey yeniden renklendirmeydi. 10'a, 8'e ve 11'e yeni renkler verdik. Bu birinci durumdu. 10, 15'in büyük atasıydı. 10'a baktık ve 10 ihlal eden elemandı. Kendi büyük atasına göre bir zikzak durumu yaratıyordu. Bu nedenle bir sağ rotasyon yaptık bunu düzelttik, yani köşelenmeyi kaldırdık ve böylece 10, 7'nin yanına geldi.

Bu üstteki şekilde. Sonra 18 yeni ihlalci oldu kendi büyük atasıyla bir uyumsuzluğu vardı. Her ikisi de aynı yöne gidiyordu. Bunu düzeltmek için bir rotasyon daha yaptık. Aslında hatırlamamız gereken tek şey de budur. Becerebiliyorsanız büyük atanızı yeniden renklendirin. Veya onu aynı hatta getirin. Sonra bir son rotasyon daha yapın ve yeniden renklendirin. Bu şekilde sistem çalışacaktır. Eğer bunları hatırlarsanız herhangi bir örneğin gerisini çözersiniz.

10'u döndürüyoruz. Bu siyah olsa iyi olur, çünkü bu durumda bu kök haline geliyor. Ama ne olursa olsun bunu siyaha döndüreceğiz çünkü orada bir siyah düğüm olması gerekiyor. Eğer aynı zamanda bunu yeniden renklendirmezsek 4. özelliği ihlal ederiz. Ben bunu niye çizmiyorum ki, aslında birkaç dakikam var. Yani eğer rotasyonu burada yapsaydık; aşağıdakinden farklı olarak B'yi alacağız.

B kırmızı. Bu size algoritmanın niye böyle olduğu ve niye başka türlü olmadığı konusunda sezgisel bir yaklaşım getirecektir. Ve C siyah. Eğer bu ağacı döndürseydik yani B'yi döndürseydik veya C'yi sağa döndürseydik elde edeceğimiz şey buydu. Bu aşağıdaki alt ağaçlar aynı şekilde sarkıyorlar. Alt ağaçlar çok iyi görünüyorlar çünkü hepsinin siyah yükseklikleri aynı. Ama gördüğünüz gibi bir problem var. Eğer B'den başlayıp aşağıdaki yaprağa gidecek olan yollara bakarsak sol taraftaki siyah düğümlerin sayısı aşağıdaki siyah yüksekliklerine bağlı.

Bunu kaydedin: siyah yüksekliği. Sağdaki tüm yolların siyah yüksekliği, C siyah olduğu için bu yükseklik artı 1 olacaktır. Şimdi biz dördüncü kuralı ihlal ettik. Bu nedenle bunu 3. Durumda yapmayız. Rotasyonu tamamladıktan sonra yeniden renklendirmeyi de yaparız. Ve bunu elde ederiz. Başka bir deyişle siyah düğümü tepeye koyduk çünkü tüm yollar o düğümden geçmek durumundadır. Oysa ki burada bazı düğümler C 'den geçiyor. Bazıları A 'dan geçiyor. Bu durum kötü. Bu

durumda biz 3. Özelliği de ihlal ederdik. Ama daha da kötüsü burada biz 4. Özelliği ihlal ediyoruz. Şimdi birazcık bunları özetleyelim.

Şu ana kadar; eğer kırmızı siyah bir ağaçta araya yerleştirme yaparsak o ağacı kırmızı siyah olarak muhafaza edebiliriz. Yani kırmızı siyah araya yerleştirme dinamik sete bir x değerini ekliyor ve bunu aynı zamanda muhafaza ediyor ve kırmızı siyahlık olgusunu koruyor. Yani kırmızı siyah ağaç olgusunu muhafaza ediyor; bu iyi çünkü bu durumda logaritmik yükseklik de korunur. Böylece kırmızı siyah ağaçlarda yapılan bütün sorgulamalar logaritmik zaman alır. Kırmızı siyah araya yerleştirmeler ne kadar zaman alır? Biliyoruz ki hazırlık zamanı için hedefimiz log n düzeyinde. Bunu resmi olarak kanıtlamayacağım ama sezgisel olarak çok açık görünüyor olması lazım. Böylece 2. Ve 3. durumlar pardon yanlış yere işaret ediyorum 2. Ve3. Durumlar son durumlardır.

3.durumu yaptığımızda işimiz biter. 2.durumu yaptığımızda 3. Durumu yapmanın başına geliriz ve ondan sonra işimiz biter. Böylece gerçekten sayma ihtiyacımız olan durum birincisidir, çünkü bütün bu yeniden renklendirme, rotasyon gibi işlemler sabit zaman alır. Bu durumda mesele bunlardan kaç tane olduğudur. 1. Durumda sadece yeniden renklendirme yapılır bu ağacı hiçbir zaman değiştirmez ve x'i sadece iki düzey yukarıya taşır.

Ağacın yüksekliğinin en çok 2 log (n+1) olduğunu biliyoruz. Yani Durum 1 lerin sayısı en çok log(n+1) olabilir. Yani durum 1'lerin sayısı en çok log n olur. Yani bunlar log n zaman alır. Ondan sonra 2.ve 3. Durumların sayısı bu kolonlardan biri için en çok 1'dir. Her ikisini bir araya getirirsek 2. Ve 3. Durumlar için en çok 2 elde ederiz. Evet sonunda log n zamanı, bu iyi. Kırmızı siyah araya yerleştirmelerin enteresan bir yönü de bunların sadece birinci düzey rotasyonları gerektirdiğidir. Yani değişikliklerin çoğu yeniden renklendirmededir. 1.durumda sadece yeniden renklendirme yapılır, rotasyon yapılmaz. 2.durumda belki bir rotasyon yapılır. Eğer 3. Duruma gerek kalırsa bir rotasyon yapılacak; böylece rotasyonların toplam sayısı 2 olur. Yani araya yerleştirme de bu 1 veya 2 tanedir.

Bu aslında hoş bir durum çünkü bir ağacı döndürmek o ağacı boyamaktan daha rahatsız edici bir durumdur. Neden? Çünkü diyelim ki bir veri yapınız var, yani bir arama ağacınız var. Teorik olarak insanlar bu arama ağacını bir şeyler için kullanmak isterler. Örneğin, burada sorgulamalar yaparlar. Mesela Google'da computer sözcüğü ile çakışan tüm dokümanlar bir arama ağacıyla temsil edilir. Şurada bir Google T-shirti var Onun için burada Google'ı referans olarak kullanabiliriz.

Bir arama ağacı var. Bu arama ağacı Google sözcüğüne ilişkin her şeyi depoluyor. Belki bir tarihten sonra değiştirilmiş olan bir şeyi burada aramak istiyorsunuz ve bu ağaçta bir takım sorgulamalar yapıyorsunuz. Ve insanlar Google'ı sorgulamalarla neredeyse yumrukluyorlar. 1 saniyede neredeyse zilyon tane sorgulama geliyor. Bu konuda sakın bana atıfta bulunmayın. Çünkü rakam doğru olmayabilir. Evet zilyon. Ama insanlar sürekli arama yapıyorlar. Ve siz

ağacı yeniden renklendirdiğinizde insanlar hala arama yapabilirler; sizin değiştirdiğiniz küçük bir parçadır. Ben arama yaptığımda bir düğümün kırmızı mı siyah mı olduğuyla ilgilenmem, çünkü her ikisinin de logaritmik yüksekliği vardır.

Böylece siz arama yapanlar web'de sörf yaparken de gelip ara sıra güncellemeler uygulayabilirsiniz. Ve bu açıdan yeniden renklendirme çok iyidir. Rotasyon ise buna göre biraz daha pahalıdır; çünkü işi yaparken önce düğümleri kilitlersiniz, onları döndürme sürecinizde kimsenin ona dokunmamasını istersiniz. Ondan sonra kilidi açarsınız. Yani rotasyon sayısının gerçekten az olması sadece 2 olması çok güzeldir. Sürenin de log n'yi geçmemesi lazım çünkü temelde sıralı bir listeye araya yerleştirme yapıyoruz. Bu durumda, eğer n sayısında araya yerleştirme yaparsak, alt sınırımız da n log n olur. Pekiyi, silme; şimdi burada bunu incelemeyeceğim.

Bunu kitaptan okumalısınız. Biraz daha karmaşıktır ama aynı fikirlere dayanır. Aynı sınırlamalara bağlıdır: log n zamanı ve 1.düzeyde rotasyonlar. Bu nedenle bunu inceleyin. Bu kırmızı siyah ağaçlardır. Şimdi artık verileri log n zamanlı hazırlık zamanında işleyebilir ve bakımını yapabilirsiniz. Bu iyi bir şeydir. İleride bunu yapmanın 3 yolunu göreceğiz.

Ders bitmiştir.