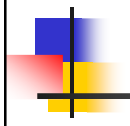


BM 307 Dosya Organizasyonu (File Organization)

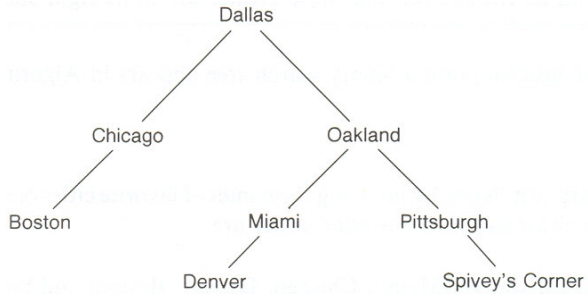
Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Binary Tree Structures
  - Binary Search Trees
  - AVL Trees
  - Internal Path Reduction Trees
  - Değerlendirme

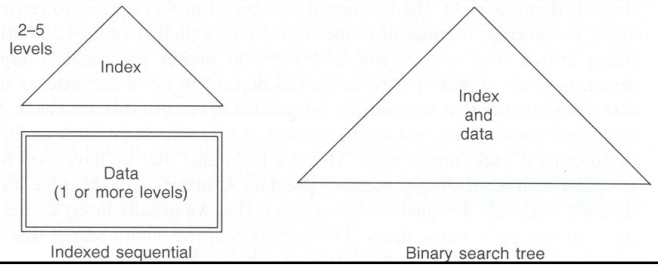
Binary Tree Structures
Binary Search Trees

- Bir binary search tree üzerindeki her node hem data saklar hemde diğer node'lara ulaşılırken yön belirler
- Bir binary search tree'nin derinliği root node'a göre maksimum seviye sahip olan node ile belirlenir
- Aşağıdaki örnekte ortalama probe sayısı 2.75' tir.
- Hash fonksiyonlarıyla oluşturulan yapılarda probe sayısı daha düşük olabilir.
- Ancak ağaç yapıları flexibility özelliğine sahiptir.
- Tüm bilgilerin küçükten büyüğe sıralı yazılabilmesi için inorder traversal yeterlidir.



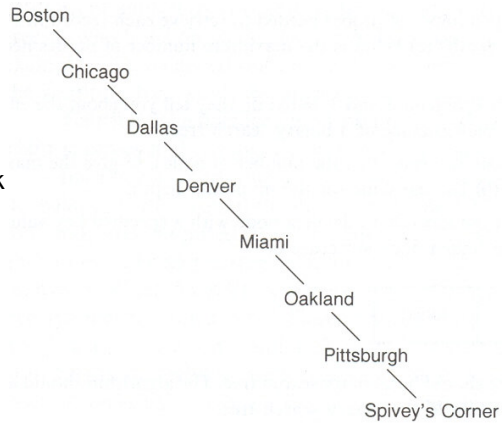
Binary Tree Structures
Binary Search Trees

- Indexed Sequential File Organization ağaç yapısını kullanır. Hem direct ve hemde sequential erişimde kullanılır.
- Indexed Sequential File Organization'da ağaç sadece indeks amacı için kullanılır ve bilgiler yapraklardadır
- Binary Search Tree'lerde ise bilgi hem yapraklarda hemde internal node'lardadır.
- Hash fonksiyonlarıyla oluşturulan yapılarda probe sayısı daha düşük olabilir.
- Saklanan bilgi arttıkça binary search tree yüksekliği artar ve performans düşer
- **Branching factor** binary seearch tree'de ikidir. Branching factor Indexed Sequential File'da daha yüksek olduğu için büyük bilgilerin saklanmasında doğrudan erişimde ve sıralı erişimde performans yüksektir.



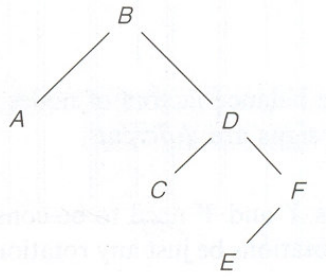
Binary Tree Structures
Binary Search Trees

- Bir binary search tree'nin yapısı bilgilerin insert edilme sırasına bağlıdır.
- Alfabetik sırayla insert edildiğinde ağacın yapısı bağlı listeye dönüşür
- Yeni oluşan ağaçta en yüksek erişim probe sayısı 8 ve ortalama probe sayısı 4.5' tir
- Önceki örnekte en yüksek erişim probe sayısı 4 ve ortalama probe sayısı 2.75'tir
- Ağaçların insert sırasında sürekli balance edilmesi performansı artırmaktadır
- AVL tree veya height-balanced tree performansı artırmak için sürekli kendisini yeniden düzenlemektedir



Binary Tree Structures
AVL Trees

- Bir AVL tree, bir binary search tree'dir ve bir node'un altağaçlarının yükseklikleri yaklaşık aynıdır
- Bir AVL tree, bilgilerin insert edilme sırasından bağımsız olarak ağacın dengeli durumunu korumasını sağlar
- Bir node'un **balance factor** değeri sağ altağacın yüksekliğinden sol altağacın yüksekliğinin çıkarılmasıyla bulunur
- Şekilde A, C, E = 0, ve F = -1, D = +1, B = +2 balance factor değerine sahiptir
- Bir AVL ağacında bütün node'ların balance factor değeri -1, 0 veya +1 olur
- Her yeni insert'ten sonra, ağaç dengesiz hale gelmişse bir veya iki döndürme işlemiyle dengelenir.

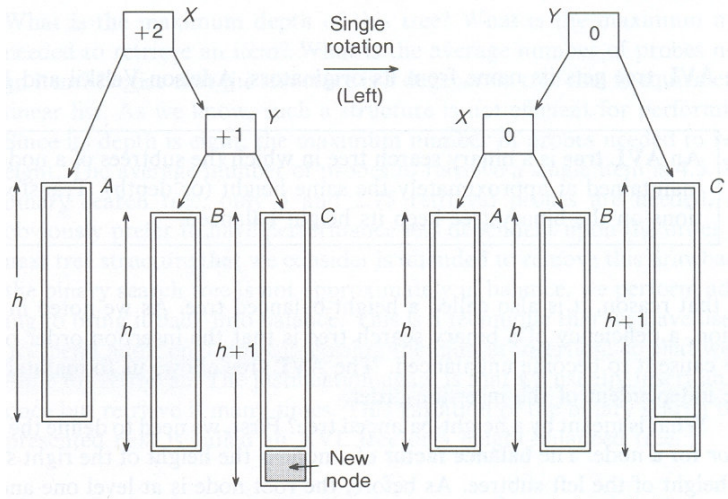


Binary Tree Structures
AVL Trees

- Bir AVL tree için döndürme işlemi temel olarak iki grupta ele alınır
- 1.durumda dengelenecek node ile child node'un balance factor değeri **aynı** işarettedir. Tek döndürme yeterlidir.
- 2.durumda ise dengelenecek node ile child node'un balance factor değeri **farklı** işarettedir. Çift döndürme gereklidir.
- Döndürme işleminden sonra **inorder** dolaşmayla elde edilecek bilgide değişme olmamalıdır

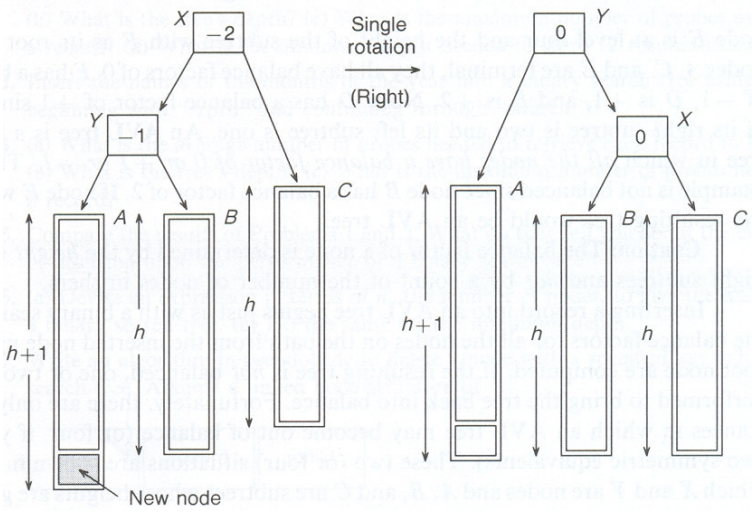
Binary Tree Structures
AVL Trees

- 1.Durum (Dengelenecek node ile child'ı arasındaki işaret aynı)



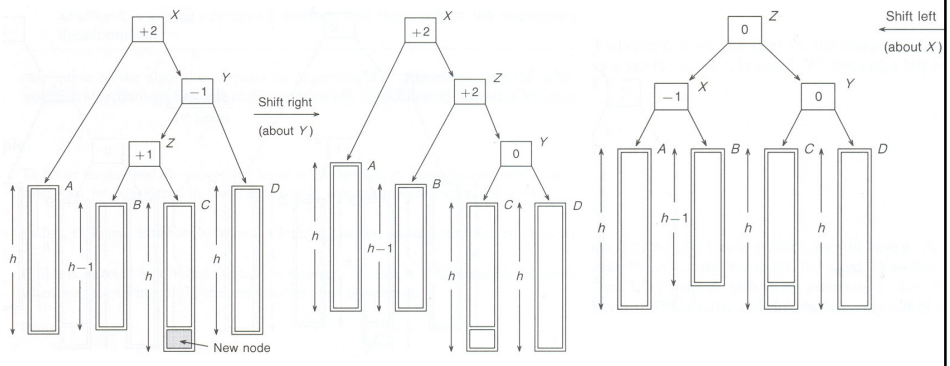
Binary Tree Structures
AVL Trees

- 1.Durum (Dengelenecek node ile child'ı arasındaki işaret aynı)



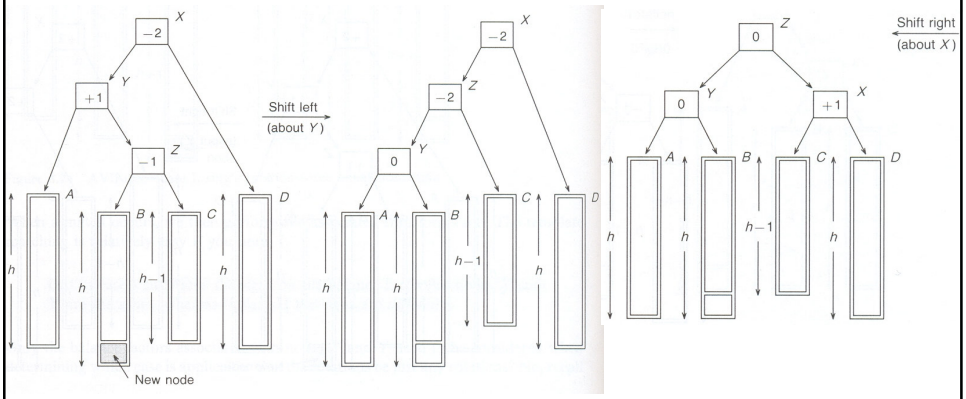
Binary Tree Structures
AVL Trees

- 2.Durum (Dengelenecek node ile child'ı arasındaki işaret farklı)



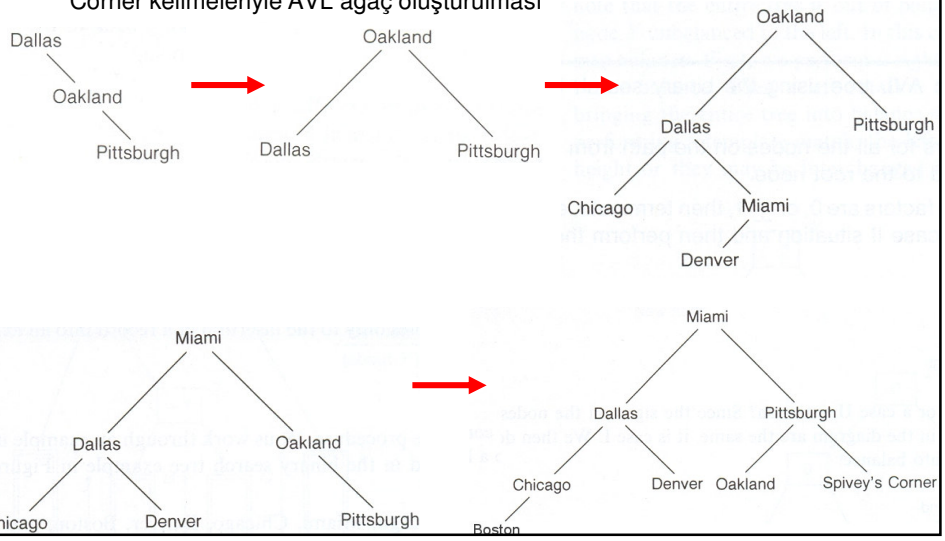
Binary Tree Structures
AVL Trees

- 2.Durum (Dengelenecek node ile child'ı arasındaki işaret farklı)



Binary Tree Structures
AVL Trees

- Örnek: Dallas, Oakland, Pittsburg, Miami, Chicago, Denver, Boston, Spivey's Corner kelimeleriyle AVL ağaç oluşturulması



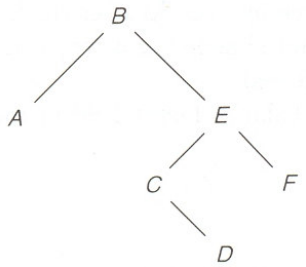
Binary Tree Structures
AVL Trees

Değerlendirme

- Tüm kayıtların eklenmesi beklenmeden her eklemeden sonra bir veya iki döndürme yapılır.
- Doğrudan erişimde worst case performansı $O(\log_2 n)$ değerinde tutulur.
- Dengesiz bir altağacın root node'unun balance factor değeri ±2 olur.
- Ekleme işlemi sırasında izlenen yol üzerinde ±1 balance factor değerine sahip olan node yoksa ağaç hala dengededir
- Eğer eklenen son yaprak kendi parent'ının ilk child'ı ise yol üzerindeki tüm node'ların balanca factor değeri yeniden düzenlenir.
- Eğer eklenen son yaprak kendi parent'ının ikinci child'ı ise sadece parent balance factor değeri yeniden düzenlenir

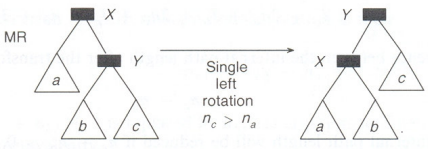
Binary Tree Structures
Internal Path Reduction Trees

- Ağaç içerisindeki tüm node'ların derinlikleri toplamını en aza indirmeyi amaçlar.
- Ağaçtaki tüm node'ların bir kez okunması için gereken toplam probe sayısına **Internal Path Length** denir.
- Bir Internal Path Reduction (IPR) Tree bir node'un altağaçlarının internal path length değerlerini dengelemeye çalışan bir Binary Search Tree'dir.
- Aşağıdaki ağaç için internal path length değeri 15'tir.



Binary Tree Structures
Internal Path Reduction Trees

- Temel olarak iki farklı durum oluşur. **MR** (sağdaki altağaçta daha fazla node var) ve **ML** (soldaki altağaçta daha fazla node var).



Internal Path Length değeri $\mathbf{n_c - n_a}$ kadar azalmıştır

- Döndürmeden önce internal path değeri, n_a , n_b, n_c , a, b, ve c altağaçlarındaki node sayısı, I_a, I_b, I_c ise a, b, c altağaçlarının internal path length değerleridir

$$I_a + n_a + I_b + 2n_b + I_c + 2n_c + 3$$

- Döndürmeden sonra internal path değeri,

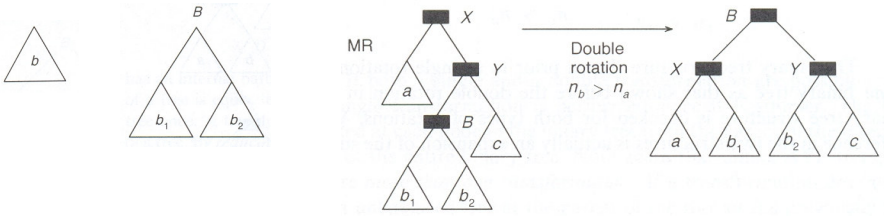
$$I_a + 2n_a + I_b + 2n_b + I_c + n_c + 3$$

- Internal path length değeri farkı,

$$n_c - n_a$$

Binary Tree Structures
Internal Path Reduction Trees

- B altağacı iki parçaya bölünürse $n_b = n_{b1} + n_{b2} + 1$ olur



- Döndürmeden önce internal path değeri

$$I_a + n_a + I_{b1} + 3n_{b1} + I_{b2} + 3n_{b2} + I_c + 2n_c + 6$$

- Döndürmeden sonra internal path değeri,

$$I_a + 2n_a + I_{b1} + 2n_{b1} + I_{b2} + 2n_{b2} + I_c + 2n_c + 5$$

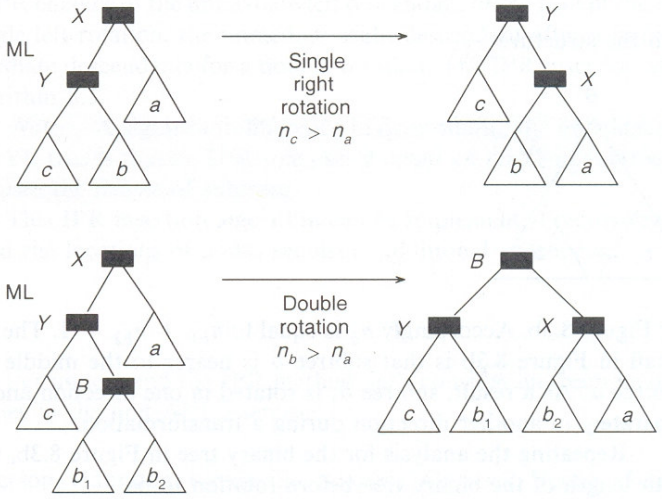
- Internal path length değeri farkı,

$$n_{b1} + n_{b2} - n_a + 1$$

- Balance kriteri $n_{b1} + n_{b2} - n_a + 1 > 0$

Binary Tree Structures
Internal Path Reduction Trees

- ML (Soldaki altağaçta daha fazla node olması durumları)

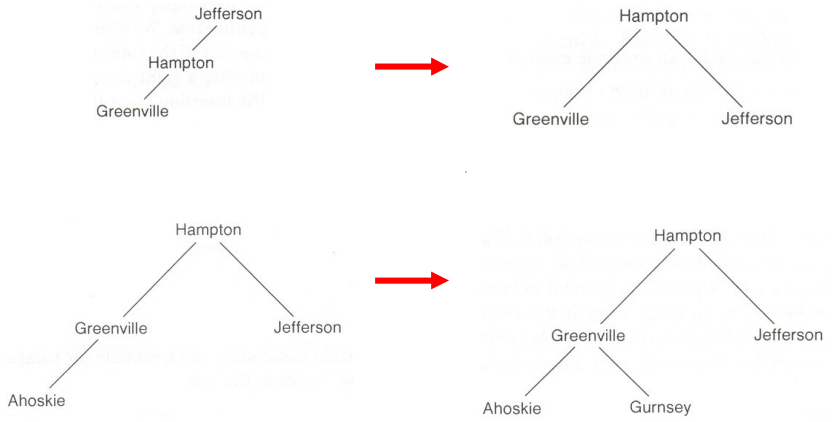


Binary Tree Structures
Internal Path Reduction Trees

- Balance kriteri her node eklemede eklenen node'un parent'ı ile root node arasındaki tüm node'lar için kontrol edilmelidir
- Bir döndürme işleminden sonra yeni oluşan altağacın eski root node'un parent'ı ile ağacın tamamının root node arasındaki tüm node'lar için kontrol işlemi yapılır
- IPR ile AVL ağaçları arasındaki en önemli fark, dengesizlik durumunda AVL ağacı altağaçların yüksekliklerini kullanır, IPR ağacı ise altağaçların node sayılarını kullanır.

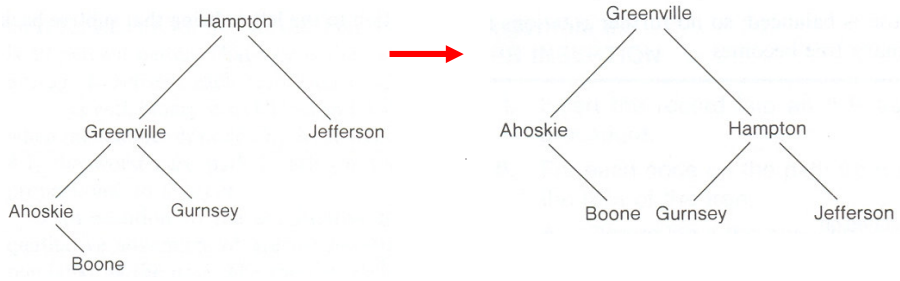
Binary Tree Structures
Internal Path Reduction Trees

- **Örnek:** Jefferson, Hampton, Greenville, Ahoskie, Gurnsey, Boone, Gusey, Hamilton kelimelerini sırasıyla insert edelim.



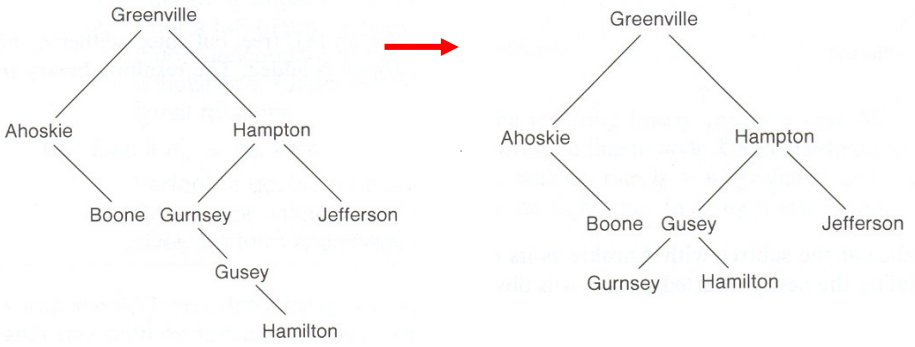
Binary Tree Structures
Internal Path Reduction Trees

Jefferson, Hampton, Greenville, Ahoskie, Gurnsey, Boone, Gusey, Hamilton



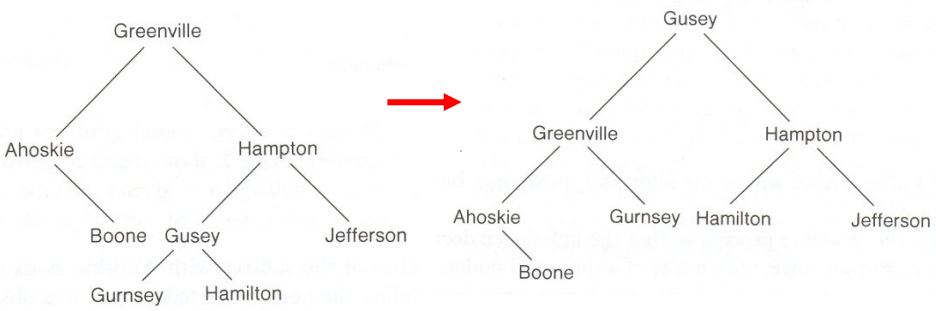
Binary Tree Structures
Internal Path Reduction Trees

Jefferson, Hampton, Greenville, Ahoskie, Gurnsey, Boone, Gusey, Hamilton



Binary Tree Structures
Internal Path Reduction Trees

Jefferson, Hampton, Greenville, Ahoskie, Gurnsey, Boone, Gusey, Hamilton



Binary Tree Structures
Internal Path Reduction Trees

Değerlendirme

- Tabloda IPR ağacının ve AVL ağacının worst case performansları görülmektedir
- IPR ağacının height değeri hiçbir zaman AVL ağacından kötü olmamaktadır.
- Ancak IPR ağacında Internal Path Length değeri aynı data için AVL ağacından %10 daha iyidir.
- Binary Search Tree ve AVL ağaçları gibi, IPR ağaçlarıda bilgi sayısı çok artınca etkin değillerdir. (çünkü worst case durumu n $\log_n$ dir)

TABLE 8.1 SUMMARY OF WORST CASE COMPLEXITIES

	AVL	IPR
Height	$\leq 1.4402 \log_2 n$	$\leq 1.4402 \log_2 n$
Internal path length	$\geq 1.2793\, n \log_2 n$	$1.0515\, n \log_2 n$

Binary Tree Structures
Internal Path Reduction Trees

Değerlendirme (Devam)

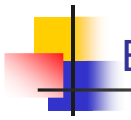
- IPR ağaçları sayısal avantajının yanında daha fazla esneklik sağlar. İzin verilecek bir denge kriter değerine göre k-balanced ağaçlar oluşturulur. İstenilen bir k değerine göre Internal Path Reduction yapılır.

$$n_c - n_a > k \qquad n_b - n_a > k$$

- Böylece k değeri değiştirilerek insertion time azaltılırken retrieval time artırılabilir.
- Dengelenecek ağacın node sayısı arttıkça elde edilecek probe sayısı iyileştirmesi ihmal edilebilir.
- Bu şekilde oluştuurlacak ağaçlara **ε-balanced** ağaçlar denir. Aşağıdaki ifadenin (döndürme kriterinin toplam node sayısına oranı) **ε** değerinden büyük olması halinde düzenleme yapılır.

$$\frac{n_c - n_a}{n_a + n_b + n_c + 2} > \epsilon$$

- IPR ağaçları AVL ağaçlarından daha kolay gerçekleştirilebilr. Balance değerini elde etmek toplam node sayısını elde etmekten daha zordur.
- IPR ağaconda her node COUNT alanı eklenir ve son eklenen node ile root arasında tüm node'ların COUNT değeri 1 artırılır.
- Ancak COUNT değerinin saklanması için gereken alan 2 bit' ten fazladır. BALANCE FACTOR değeri için 2 bit alan gereklidir.



Binary Tree Structures

Haftalık Ödev

Önceden oluşturulmuş ve dengelenmemiş bir Binary Search Tree'nin dengeli hale getirilmesi için kullanılan metodlar hakkında araştırma yapınız. Literatür taraması yaparak elde ettiğiniz makaleleri inceleyiniz. Kullanılan metodların avantajlarını ve dezavantajlarını içeren bir rapor hazırlayınız .