Algoritmalara Giriş Ekim 29, 2005
Massachusetts Institute of Technology 6.046J/18.410J
Professors Erik D. Demaine ve Charles E. Leiserson Dağıtım 18

Problem Seti 4 Çözümler

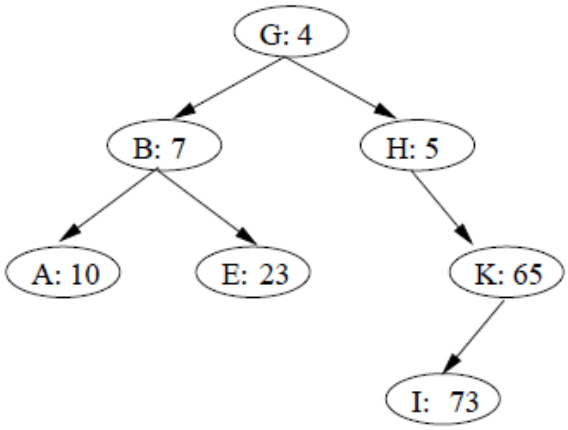
Problem 4-1. Treaps

Treap'ler (Rastgele sıralanmış ikili arama ağaçları)

Eğer tüm elemanlar aynı anda elimizde değilse - eğer elemanları birer birer alırsak, gene de bunlarla bir rastgele ikili arama ağacı oluşturabilir miyiz? Bu soruyu olumlu yanıtlayacak bir veri yapısını inceleyeceğiz. **Treap,** düğümleri değişik biçimde düzenlenmiş bir ikili arama ağacıdır. Şekil 1 bir treap örneğini göstermektedir. Genelde olduğu gibi ağaçtaki her x öğesinin bir *key[x]* anahtarı var. Buna ek olarak her x için rastgele seçilen bağımsız bir sayıyı *priority[x]* önceliği olarak atarız. Tüm önceliklerin ve tüm anahtarların farklı olduğunu kabul ederiz. Treap' in düğümleri öyle düzenlenmiştir ki, (1) anahtarların hepsi ikili-arama-ağacı özelliklerine uyarlar ve (2) özellikler de min-heap (en küçük-yığın) düzeni özelliğine uyarlar. Başka birdeyişle,

- eğer v, u' nun sol ardılıysa, key[v] <key[u];
- eğer v, u' nun sağ ardılıysa, *key[v] >key[u];* ve
- eğer v, u' nun ardılıysa, o zaman *priority (v) >priority(u)* olur.

(Özelliklerin bu karışımı nedeniyle bu ağaça "treap" denir: Hem bir ikili arama ağacının (tree), hem de bir yığının (heap) niteliklerine sahiptir.



Şekil 1: Bir treap. Her x düğümü *key[x]: priority[x]* olarak etiketlenmiş. Örneğin kökün anahtarı *G* ve önceliği 4.

Treapleri şöyle düşünmekte yarar vardır. $x_1; x_2 ,..., x_n$, düğümlerinin her birini ilgili anahtarıyla bir treap' in içine rastgele düzende yerleştirelim. Bu durumda ortaya çıkan ağaç, normal bir ikili arama ağacına (rastgele seçilmiş) öncelikleri gözetilerek belirlenen düzende düğümler araya yerleştirildiğinde oluşacak ağaçtır. Başka bir deyişle, $priority[X_i] < priority [X_j]$, X_i 'nin uygulamada X_j den önce araya yerleştirildiği anlamına gelir.

(a) Her biri farklı anahtarları ve öncelikleri olan $x_1; x_2 ,..., x_n$ seti için, bu düğümleri içeren özgün bir treap'in var olduğunu gösterin.

Çözüm:

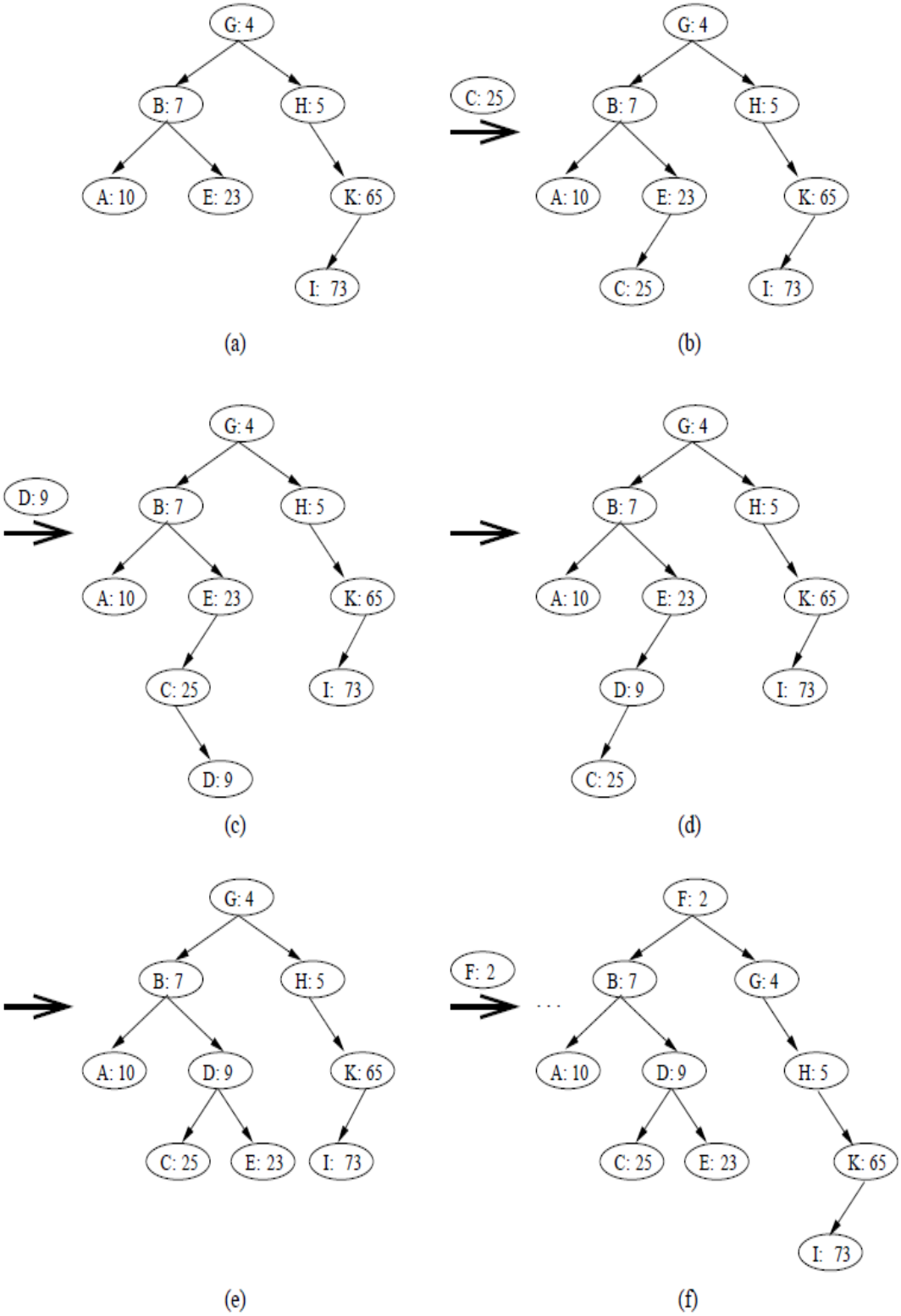
Ağaçtaki düğümlerin sayısı bağlamında tümevarımla kanıtlayın. 0 düğümü olan bir ağaçtaki taban durumu kendine özgüdür. Tümevarım açısından *k-1* ya da daha az sayıda düğümü olan treap lerin, kendine özgü olduğunu kabul edip, *k* düğümü olan bir treap' in kendine özgü olduğunu kanıtlayacağız. Bu treap' de en az öncelikli *x* öğesi kök' de olmalıdır. Sol alt-ağaçta anahtarlar < *anahtar [x]* ve sağ alt-ağaçta da anahtarlar > *anahtar [x]* olur. Bu kökü ve kökün her iki alt-ağacını da kendine özgü biçimde tanımlar. Her alt-ağaç boyutu ≤ *k-1* olan bir treap' dir, böylece bunlar tümevarımsal açıdan kendine özgüdür.

Farklı bir yolla bu, düğümlerin önemleri sırasında araya yerleştirildiği bir treap' i ele alarak da kanıtlanabilir. Tümevarım açısından en az öncelikli *k-1* düğümü olan treap' in kendine özgü olduğunu düşünün. *k=0* için bu doğrudur. Şimdi en az öncelikli *k* düğümü olan treap' i düşünün. Treap' in yapısının, anahtarların artan öncelik sırasıyla araya yerleştirildiği bir ikili arama ağacının yapısıyla aynı olduğunu bildiğimizden, en az öncelikli *k* düğümü olan treap, en az öncelikli *k-1* düğümü olan treap' e *k*' nıncı düğüm eklendiğinde aynı yapıda olur. BST (ikili arama ağacı) araya yerleştirmesi belirlenimci bir algoritma olduğundan *k*' ninci düğümün girebileceği tek yer vardır. Böylece *k* düğümü olan bir treap de kendine özgüdür ve bu tümevarımsal hipotezi kanıtlamış olur.

(b) Treap'in beklenen yüksekliğinin *O(lg n)* olduğunu ve bu nedenle treap' de bir değeri aramanın beklenen süresinin *O(lg n)* olduğunu gösterin .

Çözüm: Buradaki fikir *n* düğümü olan bir treap' in, *n* düğümü olan bir rastgele yapılanmış ikili arama ağacına eşdeğer olduğunu farketmekte yatar. Treap' de düğümlere araya yerleştirildikçe atanan öncelikler, önceliklerine göre tanımlanmış *n* düğümü, artan sırada araya yerleştirmekle aynıdır. Bu nedenle eğer öncelikleri rastgele atarsak *n* önceliğin rastgele sırasını elde ederiz ve bu da *n* giridinin rastgele permütasyonuyla aynıdır; böylece biz bunu *n* öğeyi rastgele sırada araya yerleştirme olarak görebiliriz. Bir öğeyi aramak için gerekli süre *O(h)*' dir ve *h* ağacın yüksekliğidir. Derste gördüğümüz gibi *E[h] = O(lg n).* (beklenen n düğümün permütasyonu üzerinden hesaplanır, yani önceliklerin rastgele seçimi olarak).

Mevcut bir treap' de yeni bir *x* düğümünün nasıl araya yerleştirileceğini görelim. Yapacağımız ilk şey *x*' e bir rastgele öncelik, *priority[x]* atamaktır. Sonra TREAP-INSERT dediğimiz araya yerleştirme algoritmasını çağırırız ve bunun işlemesi şekil 2' de gösterilmiştir.



Şekil 2: TREAP-INSERT' ün çalışması. Şekil 1' de olduğu gibi her *x* düğümü key[x] : *priority[x]* olarak etiketlenmiştir.(a), araya yerleştirme öncesi orijinal treap. (b), anahtarı C ve önceliği 25 olan bir düğümün araya yerleştirilmesi sonrası treap. (c) ve (d), anahtarı D ve önceliği 9 olan bir düğümün araya yerleştirilmesindeki ara evreler. (e), (c) ve (d)' deki yerleştirilmeler tamamlandıktan sonraki treap. (f), anahtarı F ve önceliği 2 olan bir düğümün araya yerleştirilmesi sonrasında treap.

(c) TREAP-INSERT' ün nasıl çalıştığını açıklayın. Fikri İngilizce açıklayın ve sözde kodunu yazın. (İpucu: Normal ikili arama ağacı araya yerleştirmesi yapın ve sonra rotasyon yani dönmelerle min-heap (en-küçük yığın) düzeni özelliğini yeniden geri getirin.

Çözüm: İpucu fikri açıklıyor: Önce normal ikili arama ağacı araya yerleştirmesini yapın ve sonra döndürmeleri uygulayarak en-küçük yığın düzeni özelliğini yeniden oluşturun.

TREAP-INSERT *(T, x)* treap *T*' de (*T*' yi değiştirerek) *x*' i araya yerleştirir. Bunun için *x*' in tanımlı *key*(anahtar) ve *priority* (öncelik) değerleri olmalıdır. Kitapta tanımlandığı şekilde TREE-INSERT, RIGHT-ROTATE ve RIGHT-ROTATE alt yordamlarını kullandık.

```
TREAP-INSERT(T, x)
1  TREE-INSERT(T, x)
2  while x ≠ root[T] and priority[x] < priority[p[x]]
3       do if x = left[p[x]]
4            then RIGHT-ROTATE(T, p[x])
5            else  LEFT-ROTATE(T, p[x])
```

Ata işaretleyicilerinin kodu basitleştirdiğine, ancak gerekli olmadığına da dikkat edin. Kökten *x'* e kadar olan yol üzerinde (TREE-INSERT çağrısından sonra), sadece her düğümün atasını bilmemiz yeterli olduğundan, bunların izini kendimiz sürebiliriz.

(d) TREAP-ARAYA YERLEŞTİRMESİ'NİN (TREAP-INSERT) beklenen koşma süresinin O(lg n) olduğunu gösterin.

Çözüm:

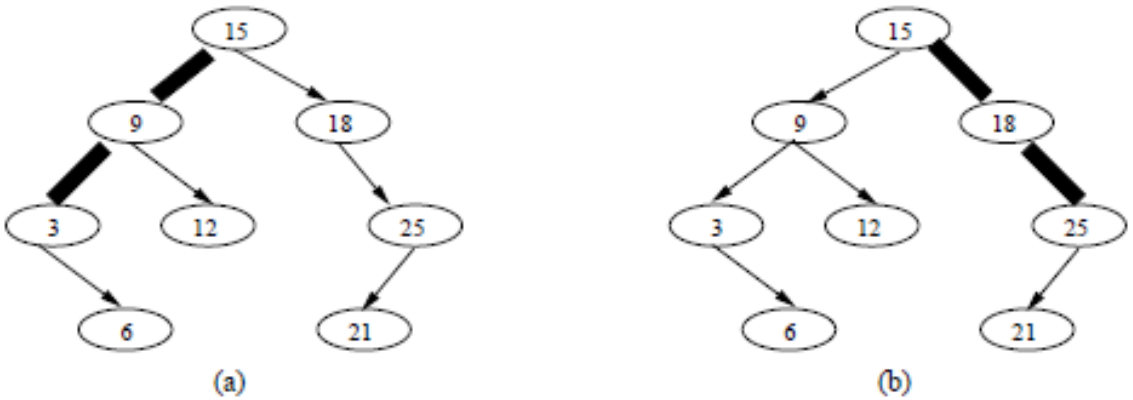
TREAP-INSERT, ağaca, bir elemanı önce normal ikili arama ağacı araya yerleştirmesi kullanarak ekler ve sonra da en küçük yığın özelliğini tekrar yapılandırmak için bir takım döndürmeler uygular.

Normal ikili arama ağacı araya yerleştirme algoritması olan TREE-INSERT, her zaman yeni öğeyi ağacın yeni bir yaprağına yerleştirir. Bu nedenle daha önce derste gördüğümüz gibi, bir treap' de bir öğeyi araya yerleştirme için gereken süre, rastgele yapılanmış bir ikili arama ağacının yüksekliğine orantılıdır ve bekleneni *O(lg n)*' dir.

Döndürme sayısının en çok olduğu durum yeni öğenin önceliğinin ağaçtaki tüm önceliklerden daha az olduğu durumdur. Bu durumda döndürmenin, bir yaprakdan köke doğru yapılması gerekir. Döndürme sayısındaki üst sınır, bu yüzden rastgele yapılanmış ikili arama ağacının yüksekliğidir ve bunun da bekleneni *O(lg n)*' dir. (bunun oldukça gevşek bir sınır olduğunu göreceğiz). Her döndürme sabit zaman aldığından dönme işleminin beklenen süresi *O(lg n)*' dir.

Böylece TREAP-INSERT ün beklenen koşma süresi *O(lg n + lg n) = O(lg n)' dir* .

TREAP-INSERT bir arama ve ardından da bir dizi döndürme yapar. Arama ve döndürmenin asimptotik koşma süreleri aynı olmakla birlikte, pratikte maliyetleri farklıdır. Bir arama, treap' deki bilgiyi değiştirmeden okurken, bir rotasyon, treap' in içindeki ata ve ardıl işaretleyicilerini değiştirir. Birçok bilgisayarda okuma işlemleri, yazma işlemlerinden çok daha hızlıdır. Bu nedenle TREAP-INSERT ' ün az döndürme yapmasını isteriz. Beklenen döndürme sayısının bir sabitle sıralı olduğunu göstereceğiz (aslında 2' den az)!



Şekil 3: Bir ikili arama ağacının omurgaları. Sol omurga **(a)**' da kalın çizgiyle işaretlenmiş, sağ omurga da **(b)**' de kalın çizgiyle işaretlenmiş.

Bu özelliği gösterebilmek için şekil(3)' de açıklanan bazı tanımlara ihtiyacımız var. Bir *T* ikili arama ağacının ***sol omurgası***, kökten en küçük anahtara giden yoldur. Başka bir deyimle sol omurga kökten çıkan ve sadece sol kenarları içeren en uzun yoldur. Simetrik olarak *T* ağacının sağ omurgası kökten çıkan ve sadece sağ kenarları içeren en uzun yoldur. Bir omurganın uzunluğu, içerdiği düğümlerin sayısıdır.

(e) TREAP-INSERT kullanılarak x' in araya yerleştirildiği anın sonrasında T Treap' ini düşünün. C, x' in sol alt-ağacının sağ omurgasının uzunluğu olsun. D de, x' in sağ alt-ağacının sol omurgasının uzunluğu olsun. x' in araya yerleştirilmesinde yapılan döndürmelerin toplam sayısının C + D' ye eşit olduğunu gösterin.

Çözüm: Döndürmelerin sayısı konusundaki iddiayı tümevarım kullanarak yapın. Taban durumu, *x*' in, *y*' nin atası olduğu durumdur. Döndürme, *y*' nin yeni kök olacağı şekilde yapıldığında *y*' nin tek ardılı olur, yani *C* + *D* = 1.

Tümevarım yaklaşımıyla *x*' in araya yerleştirilmesi sırasında yapılan döndürmelerin sayısı olan *k*, *C+D* ' ye eşittir. Taban durumu *x*' in bir yaprak olarak araya yerleştirilmesi ve böylece 0 döndürmeye ihtiyaç olmasıdır. Bu durumda *C* + *D* = *0*. Tümevarım adımını kanıtlamak için *k-1* döndürmeden sonra *C+D* = *k-1* ise, *k* döndürme sonrasında *C+D=k* olduğunu gösteririz. Sol ve sağ döndürmelerin bir şeklini çizin ve iki durumda da *C+D*' nin 1 arttığını gözleyin. *y, x'* in atası olsun ve *x*' in *y*'nin sol ardılı olduğunu varsayın. Bir sağa döndürme yaptıktan sonra *y, x'* in sağ ardılı olur ve *x*' in önceki sağ ardılı *y'* nin sol ardılı olur. Yani döndürme öncesinde *x*' in sağ alt-ağacının sol omurgası *y*' ye takılır. Böylece o omurganın uzunluğun bir artar. *x'* in sol alt ağacı sağ döndürmeden etkilenmez. Sola döndürme durumu bunun simetriğidir. Böylece bir döndürme sonrasında, *C+D* bir artar ve *k=C+D* olur ve tümevarım hipotezi kanıtlanır.

Şimdi C ve D' nin beklenen değerlerini hesaplayacağız. Basit olması için anahtarların 1, 2,..., *n* olduğunu varsayacağız. Bu varsayım, bizi genellikten uzaklaştırmaz çünkü sadece anahtarları karşılaştırıyoruz.

İki ayrı düğüm *x* ve *y* için *k* = *key[x]* ve *i* = *key[y]* olsun ve göstergesel rastgele değişkeni şöyle tanımlayalım:

$$X_{i,k} = \begin{cases} 1 & \text{eğer y, x' in sol alt-ağacının sağ omurgasının düğümü ise } (T' \text{de}). \\ 0 & \text{aksi takdirde.} \end{cases}$$

(f) $X_{i,k}$ = 1, eğer ve sadece eğer, (if and only if)

(1) *priority[y] > priority[x],*

(2) *key[y] < key[x],* ve

(3) her *z* için *key[y] < key[z] < key[x],* olursa *priority[y] < priority[z]* olacağını gösterin.

Çözüm:

Bu savı kanıtlamak için ‘’eğer ve sadece eğer’’ in her iki yönünü de kanıtlamamız gerekir. Önce eğer yönünü kanıtlayacağız. Burada eğer (1) *priority[y] > priority[x],* (2) *key[y] < key[x],* ve (3) her *z* için *key[y] < key[z] < key[x]* doğruysa *priority[y] < priority[z] olduğundan,* $X_{i,k}$ = *1*' in doğru olduğunu kanıtlayacağız. Bunu çelişki yöntemiyle kanıtlayacağız. $X_{i,k}$ = 0 olduğunu varsayın yani *y*' nin,

x' in sol alt ağacının sağ omurgasından olmadığını farzedin. Bunun bir çelişki olduğunu göstereceğiz. Eğer y, x' in sol alt ağacının sağ omurgasında değilse şu üç yerden birinde olabilir:

1. y' nin x' in sağ alt ağacında olduğunu farzedin. Bu koşul (2) ile çelişir çünkü *key[y] < key[x]' tir.*

2. *y'* nin *x'* in alt ağaçlarından birinde olmadığını farzedin. O zaman *x* ve *y*' nin *z* gibi bir ortak atası olmalıdır. *key[y] < key[x]* olduğundan *y*' nin *z'* nin sol alt ağacında ve *x*' in de *z*' nin sağ alt ağacında olduğunu biliyoruz ve *key[y] < key[z] < key[x]* durumu var. *y*, ağaçta *z'* nin altında olduğundan *priority[z] < priority[x]* ve*priority[z] < priority[y]* durumu oluşur. Bu da koşul(3) ile çelişir.

3. *y*' nin *x*' in sol alt ağacında olduğunu ama x' in sol alt ağacının sağ omurgasında olmadığını farzedin. Bunun anlamı *x*' in sol alt ağacında, *y*' nin *z* gibi bir atasının varolduğudur ve *y , z*'nin sol altağacında olur. Böylece *key[y] < key[z] < key[x]'dir. z, y'* nin atası olduğundan, *priority[z] < priority[y]* olur ve bu da koşul(3) ile çelişir.

Tüm olası durumlar çelişkilere yol açtığından $X_{i,k}$ = 1'dir.

Şimdi ''sadece eğer'' kısmına gelelim. Eğer $X_{i,k}$ = 1 ise (1), (2), ve (3). koşulların doğru olduğunu kanıtlarız. $X_{i,k}$ = 1 ise, *y, x*' in sol alt ağacının sağ omurgasındadır. *y, x'* in bir alt ağacında olduğundan, *y*' nin *x'* den sonra araya yerleştirilmiş olması gerekir, böylece *priority[y] > priority[x]* olur ve bu Koşul(1)' i kanıtlar. *y, x'* in sol alt ağacında olduğundan *key[y] < key[x],* Koşul (2)' yi kanıtlar. Koşul(3)' ü çelişki yöntemiyle kanıtlarız: $X_{i,k}$ = 1olduğunu ve *key[y] < key[z] < key[x]* ile *priority[z] < priority[y]*' yi sağlayan bir *z* olduğunu farzedin. Başka bir deyişle, *z, y' den önce* araya yerleştirilmişti. *key[z] < key[x]* koşulunu karşılayan 3 olası durum vardır:

1. *z*' nin *x*' in sol alt ağacının sağ omurgasında olduğunu varsayın. *y*' nin, *x*' in sol alt ağacının sağ omurgasında araya yerleştirilmesi için *z*' nin sağ alt ağacında araya yerleştirilmesi gerekir. *key[y] < key[z]* olduğundan bu bir çelişkiye yol açar.

2. *z*' nin *x*' in sol alt ağacında olduğunu ancak sağ omurgasında olmadığını varsayın. Bunun anlamı *z'* nin, *x*' in sağ omurgasındaki bir *z'* düğümünün sol alt ağacında olduğudur. Bu nedenle *y*' nin *x*' in sol alt ağacının sağ omurgasında araya yerleştirilmesi için, *[z']* nün sağ alt ağacında araya yerleştirilmesi gerekir bu da Koşul(1)' dekine benzer mantıkla bir çelişkiye yol açar.

3. *z*' nin, *x*' in alt ağaçlarından birinde olmadığını farzedin. Öyleyse *z* ve *x*'in, *[z']* gibi bir atası vardır ve *z*, *[z']* ' nün sol alt ağacında, *x* de sağ alt ağacındadır. Bunun anlamı *key[z] < key[z'] < key[x]* dir. *key[y] < key[z] < key[z']* olduğundan *y*, *[z']* 'nün sağ alt ağacında araya yerleştirilemez. Yani *x*' in bir alt ağacında araya yerleştirilemez ve bu bir çelişkidir.

Sonuçta *key[y] < key[z] < key[x]* ve *priority[z] < priority[y]* olan bir *z* olamaz. Bu Koşul(3)' ü kanıtlar. Böylece biz hem ''eğer'' hem de ''sadece eğer'' durumlarını her iki yönde kanıtlayarak savımızı kanıtlamış oluruz.

(g) Şunu gösterin:

$$\Pr\{X_{i,k} = 1\} = \frac{(k-i-1)!}{(k-i+1)!} = \frac{1}{(k-i+1)(k-i)}.$$

Çözüm: Bir önceki şıkta $X_{i,k} = 1$ durumunun, *y* ile *x* arasındaki öğelerin öncelikleri eğer ve sadece eğer belli bir sıradaysa gerçekleşebildiğini gösterdik. Tüm sıralamalar eşit olasılıklı olduğundan, istenen sıraya uyan önceliklerin permütasyonlarının sayısını sayarız ve öncelik permütasyonlarının toplam sayısına böleriz.

"(e)" şıkkında, $X_{i,k}$ = 1 olup olmamasının, *y* ve *x*' in önceliklerinin bağıl sıralamasına ve *key[y] < key[z] < key[x]* durumunu karşılayan tüm *z*' lere bağımlı olduğunu kanıtlamıştık. Elemanların anahtarlarının *{1,..., n}*' den türetildiğini farzettiğimizden bu durumdaki elemanların anahtarları *i, i + 1, i + 2,..., k - 1, k* ' dır. Bu elemanların önceliklerinin *(k - i + 1)!* permütasyonu vardır. Bu permütasyonların $X_{i,k} = 1$ için olanlarında, *i* en az önceliğe, *k* ikinci en az önceliğe sahiptir; kalan *(k - i - 1)* eleman da herhangi bir sıradaki önceliklere sahiptir. Bu permütasyonlardan *(k - i + 1)!* tane vardır. Böylece ''kötü'' bir düzen elde etmemizin olasılığı *(k – i - 1)! / (k - i+ 1)! = 1 / (k - i)(k – i + 1)* olur.

(h) Şunu gösterin;

$$\mathrm{E}[C] = \sum_{j=1}^{k-1} \frac{1}{j(j+1)} = 1 - \frac{1}{k}.$$

Çözüm:

Anahtarı *k* olan bir *x* düğümü için *x*' in sol alt ağacının sağ omurgasındaki düğümlerin beklenen sayısı *E [C]'* dir. bu ağaçtaki tüm i değerleri için $X_{i,k}$ rastgele değişkenlerinin beklenen değerinin toplamına eşittir. Tüm $i \geq k$ düğümlerinde $X_{i,k} = 0$ olduğundan, sadece i<k durumunu ele almamız yeterli olur.

$$\begin{aligned}
\mathrm{E}[C] &= \sum_{i=1}^{k-1} \mathrm{E}[X_{i,k}] = \mathrm{E}\left[\sum_{i=1}^{k-1} X_{i,k}\right] \\
&= \sum_{i=1}^{k-1} \Pr\{X_{i,k} = 1\} \\
&= \sum_{i=1}^{k-1} \frac{1}{(k-i)(k-i+1)} \\
&= \sum_{j=1}^{k-1} \frac{1}{j(j+1)}
\end{aligned}$$

Bu toplamı basitleştirmek için;

$$\frac{1}{j(j+1)} = \frac{j+1-j}{j(j+1)} = \frac{1}{j} - \frac{1}{j+1}.$$

olduğuna dikkat edin. Böylece toplamlar birbirini götürür. Ve elimizde şu kalır:

$$\mathrm{E}[C] = 1 - \frac{1}{k}.$$

Eğer bunu göremediyeniz, aşağıdaki denklemin *k* üzerinde tüme varımla geçerli olduğunu kanıtlayabilirdiniz.

$$\sum_{j=1}^{k-1} \frac{1}{j(j+1)} = 1 - \frac{1}{k}$$

Tümevarım hipotezini kanıtlama sırasında

$$\frac{1}{j} - \frac{1}{j+1} = \frac{1}{j(j+1)}.$$

olduğunu keşfedebilirdiniz.

(i) Aşağıdakini göstermek için bir simetri argümanı kullanın:

$$\mathrm{E}[D] = 1 - \frac{1}{n-k+1}.$$

Çözüm: buradaki fikir, anahtarlar arasındaki sıralama bağlantısı değişirse, oluşan treap' i düşünmekten geçer. Yani tüm *x* elemanları için, *priority[x]*' i değiştirmeden *key[x]'* i, *n - key[x]* + 1 ile değiştirin.

Başlangıçtaki anahtarları kullanarak elemanları yerleştirdiğimizde (önceliklerin artan sırasıyla) elde ettiğimiz ikili ağaç *T* olsun. Anahtarları yeniden eşlemleyip yeni bir ikili ağaçta araya yerleştirdiğimizde, şekli *T*' nin şeklinin ayna kopyası olan bir *[T']* ağacı elde ederiz; (soldan-sağa yansıtılmış). *T*' de anahtarı *k* olan ve bu nedenle de anahtarı *[T']* nde *n - k + 1* olan *x* elemanını düşünün. *x*' in *T*' deki alt ağacındaki sol omurgasının uzunluğu, *x*' in *[T']*' ndeki sol alt ağacının sağ omurgasının uzunluğu olur. Şık(g)' den, *y* düğümünün sol alt ağacının sağ omurgasındaki beklenen uzunluğunun *1 – 1 / idkey[y]* olduğunu biliyoruz; böylece *x*' in *[T']*' ndeki sol alt ağacının sağ omurgasının beklenen uzunluğu *1 - 1/(n - k + 1)* olur. Bu şu anlama gelir:

$$\mathrm{E}[D] = 1 - \frac{1}{n-k+1}.$$

(j) Bir treap' e bir düğüm eklenirken yapılan döndürmelerin beklenen sayısının 2' den az olduğu sonucunu çıkarın.

Çözüm:

E [döndürme sayısı]

$$\begin{aligned} &= \mathrm{E}[C+D] = \mathrm{E}[C] + \mathrm{E}[D] = 1 - \frac{1}{k} + 1 - \frac{1}{n-k+1} \\ &\leq 1+1 = 2. \end{aligned}$$

Problem 4-2. Dengeli olmak

Bir ağaç ailesine, ailedeki her ağacın boyu O(lg n) ise ***dengeli*** deyin; burada n, ağaçtaki düğümlerin sayısıdır. (Ağacın *yüksekliğinin* ağacın kökünden yaprağına kadar olan herhangi bir yoldaki kenar sayılarının maksimumu olduğunu hatırlayın. Özellikle tek düğümü olan bir ağacın yüksekliği 0 olur.)

Aşağıdaki her özellik için ikili ağaçlar ailesinin dengeli olma özelliğini taşıyıp taşımadığını belirleyin. Eğer yanıtınız "hayır" ise, bir zıt örnek verin. Eğer yanıtınız "evet" ise kanıtlayın; (ipucu: Kanıtlama tümevarımla yapılmalıdır). Dengeli olmanın *asimptotik* bir özellik olduğunu hatırlayın; bu nedenle zıt örnekleriniz, sadece tek ağacı tanımlamamalı, ailedeki sonsuz sayıda sete yönelik olmalıdır.

(a) Ağacın her düğümü ya bir yapraktır ya da 2 ardılı vardır.

Çözüm: Hayır. Bunun ters-örneği her düğümünden sola doğru bir yaprağın asılı olduğu bir sağ zincirdir.

(b) Her alt ağacın boyutu $2^k - 1$ olarak yazılabilir; burada k bir tamsayıdır (her alt ağaç için aynı *değildir).*

Çözüm: Evet.

Kökü r olan herhangi bir alt-ağaç düşünün. Koşuldan dolayı bu alt ağacın boyutunun $2^{k_1} - 1$ olduğunu biliyoruz. Ayrıca *r'* nin sol ardılından köklenen alt-ağacın boyutunun $2^{k_2} - 1$ ve r' nin sağ ardılından köklenen alt-ağacın boyutunun da $2^{k_3} - 1$ olduğunu biliyoruz. Ama *r'* den köklenen alt-ağacın boyutu basitçe r düğümü artı sol ve sağ alt ağaçların düğümleridir. Bize $2^{k_1} - 1 = 1 + (2^{k_2} - 1) + (2^{k_3} - 1)$, ya da $2^{k_1} = 2^{k_2} + 2^{k_3}$ denklemini verir.

Şimdi $k_2 = k_3$ olduğunu göstereceğiz. Bunu görmek k_1, k_2 ve k_3 'ün ikili gösterimlerini düşündüğümüzde kolaydır.

Aksi halde $k_2 \leq k_3$ olduğunu mantık kullanarak (WLOG) farzedersek o durumda, $2^{k_1 - k_2} = 1 + 2^{k_3 - k_2}$ olur. Şimdi bu denklemi karşılayacak ve 2' nin katları olan sadece 2 tamsayı vardır. Ve bunlar 2^1 ve 2^0' dır. Böylece $k_3 - k_2 = 0$ veya $k_2 = k_3$ olur ve *r*' nin sol ve sağ alt ağaçlarında aynı sayıda düğüm vardır. Bu, ağacın mükemmel dengede olması demektir.

(c) Öyle bir $c > 0$ sabiti vardır ki, ağaçtaki her düğüm için, o düğümden çıkan daha küçük alt- ağacın boyutu, daha büyük alt-ağacın boyutunun en az *c* ile çarpımına eşittir.

Çözüm: Evet[1].

Kanıtlama tümevarımla yapılır. İki alt –ağaçında *n* düğümü olan *x*' in, *y* ve *z* gibi iki ardılı olduğunu ve bunların da alt-ağaçlarındaki düğümlerin boyutunun n_1 ve n_2 olduğunu farz edin. Tümevarım hipotezi gereği belli bir *d* sabiti için , *y*' nin alt-ağacının yüksekliği en çok $d \lg n_1$ ve *z*' nin alt-ağacının yüksekliği de en çok $d \lg n_2$' dir. Şimdi *x'* in alt-ağacının yüksekliğinin en çok $d \lg n$ olduğunu kanıtlayalım. Mantıksal yaklaşımla $n_1 \geq n_2$ olduğunu varsayın. Böylece problemin söyleminden $n_2 \geq cn_1$ elde ederiz. Bu nedenle $d \lg(1 + c) \geq 1$ ise, $n = n_1 + n_2 + 1 \geq (1 + c) n_1 + 1 \geq (1 + c)n_1$ ve *x*' in alt-ağacının yüksekliği olan *h* için;

$d \lg n_1 + 1 \leq d \lg (n / (c + 1)) + 1 \leq d \lg n - d \lg(1 + c) + 1 \leq d \lg n$ elde ederiz. Sonuçta yeterince büyük *d* için *n* sayıda düğümü olan bir ağacın yüksekliği en fazla $d \lg n$' dir.

(d) Öyle bir *c* sabiti vardır ki, ağaçtaki her düğüm için, o düğümün ardıl alt-ağaçlarının yükseklikleri arasındaki fark en çok *c* 'dir.

Çözüm: Evet[1]. $n(h)$, *h* yüksekliğinde olan ve belirtilen özelliği karşılayabilen bir ağacın düğümlerinin asgari sayısı olsun. $0 < \alpha \leq 1$ olan bir sabit için $n(h) \geq (1 + \alpha)^h - 1$ olduğunu tümevarımla gösteririz. Sonra n düğümü olan bir ağaçta $h \leq \log_{1+\alpha}(n+1) = O(\lg n)$ sonucuna varırız.

Taban durumunda yüksekliği 0 olan bir ağacın tek düğümü vardır ve $\alpha \leq 1$ için, $1 \geq (1 + \alpha)^0 - 1$ olur.

Tümevarım adımında yüksekliği $k < h$ olan tüm ağaçlarda $n(k) \geq (1+\alpha)^k - 1$ olduğunu varsayın. Şimdi yüksekliği *h* olan bir ağaç düşünün ve iki alt-ağacına bakın. Bir alt-ağacın yüksekliğinin $h-1$ olduğunu biliyoruz ve diğerinin de yüksekliğinin en az $h-1-c$ olması gerekir. Bundan şu ifade çıkar:

$$n(h) \geq n(h - 1) + n(h - 1 - c) + 1.$$

[1] Bu problemde bir 0 işaretçisinin, boyutu 0 olan bir alt-ağaç olduğunu varsayıyoruz, yani tek ardılı olan bir düğümün iki alt-ağacı var ve bunlardan birinin boyutu 0. Eğer siz tek ardılı olan bir düğümün tek alt-ağacı olduğunu varsayarsanız, o zaman bu problem yanıtınız ''hayır''olmalıdır.

Tümevarım hipotezini kullanarak şu ifadeye ulaşırız:

$$\begin{aligned} n(h) &\geq (1+\alpha)^{h-1} - 1 + (1+\alpha)^{h-1-c} - 1 + 1 \\ &\geq 2(1+\alpha)^{h-1-c} - 1. \end{aligned}$$

Alfayı yeterince küçük seçtiğimizi ve $(1 + \alpha) < 2^{1/(c+1)}$ olduğunu farz edin. O zaman $(1 + \alpha)^{c+1} < 2$ elde ederiz. Sonuçta aşağıdaki eşitsizliğe ulaşırız;

$$n(h) \geq 2(1+\alpha)^{h-1-c} - 1 \geq (1+\alpha)^{h} - 1.$$

Sonuçta tümevarım hipotezini karşılamış oluruz.

Dikkat ederseniz α' nın bu değerine $h \leq \log_{1+\alpha}(n + 1)$ içine tekrar yerleştirirsek şu eşitsizliği elde ederiz;

$$h \leq \frac{\lg(n+1)}{\lg(1+2^{c+1})} \leq (c+1)\lg(n+1).$$

(e) Bir düğümün ortalama derinliği *O(lg n)*'dir. (Hatırlarsanız, bir *x* düğümünün derinliği, ağacın kökünden *x* düğümüne kadar olan yoldaki kenarların sayısıydı.)

Çözüm: Hayır.

Bir ağaç düşünün: $n - \sqrt{n}$ düğümü tam bir ikili ağaç biçiminde düzenlenmiş olsun ve diğer $\sqrt{n}$ düğümü de dengeli ağaçta $\sqrt{n}$ uzunluğunda bir zincir gibi uzuyor olsun. Bu ağacın yüksekliği $\lg(n - \sqrt{n}) + \sqrt{n} = \Omega(\sqrt{n})$ iken bir düğümün ortalama derinliği en çok:

$$\begin{aligned} & (1/n)\left(\sqrt{n}\cdot(\sqrt{n}+\lg n) + (n-\sqrt{n})\cdot\lg n\right) \\ = \; & (1/n)(n + \sqrt{n}\lg n + n\lg n - \sqrt{n}lgn) \\ = \; & (1/n)(n + n\lg n) \\ = \; & O(\lg n) \end{aligned}$$

olur. Böylece ortalama düğüm derinliği *O(lg n)* ama yüksekliği $\Omega(\sqrt{n})$ olan bir ağacımız olur.