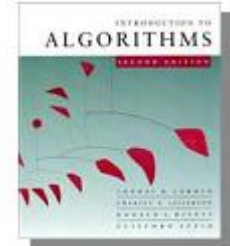


6.Hafta

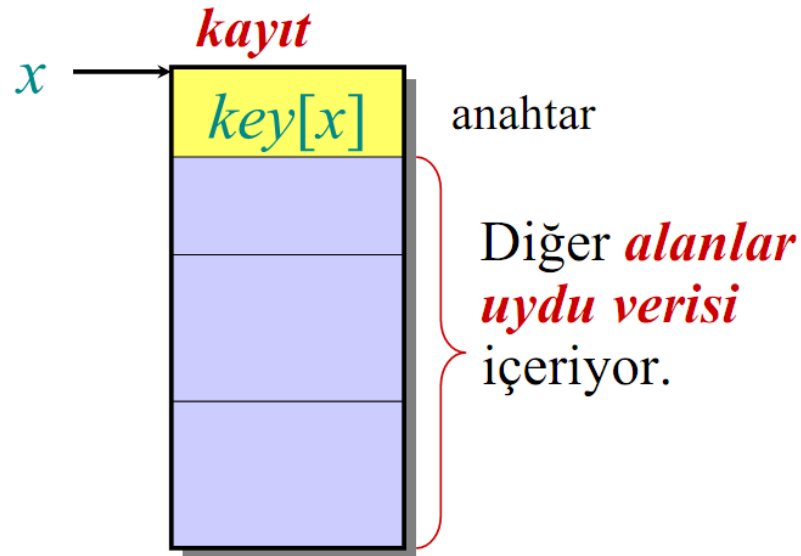
Kıyım Fonksiyonu (Hashing), BST

- Doğrudan erişim tabloları
- Çarpışmaları ilmekleme ile çözmek
- Kıyım fonksiyonu seçimi
- Açık adresleme

Sembol-tablosu problemi



Sembol tablosu S 'nin içinde n *kayıt* var:

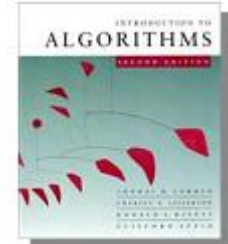


S 'deki işlemler:

- ARAYA SOKMA(S, x)
- SİLME(S, x)
- ARAMA(S, k)

Veri yapısı S nasıl organize edilmelidir?

Doğrudan erişim tablosu



FİKİR: Anahtarların $U \subseteq \{0, 1, \dots, m-1\}$ setinden seçildiğini ve birbirlerinden farklı olduklarını varsayın. $T[0 \dots m-1]$ dizilimini oluşturun:

$$T[k] = \begin{cases} x & \text{eğer } x \in K \text{ ve } \text{key}[x] = k \text{ ise,} \\ 0 & \text{diğer durumlarda.} \end{cases}$$

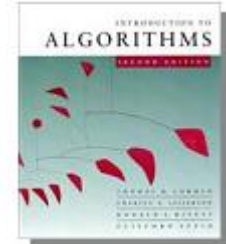
Burada işlemler $\Theta(1)$ zamanı alır.

Problem: Anahtarların değer kümesi büyük olabilir:

- 64-bit sayılar (18,446,744,073,709,551,616 farklı anahtarları temsil eder),
- (daha da fazla) karakter dizgisini içerebilir.

Çözüm HASHING

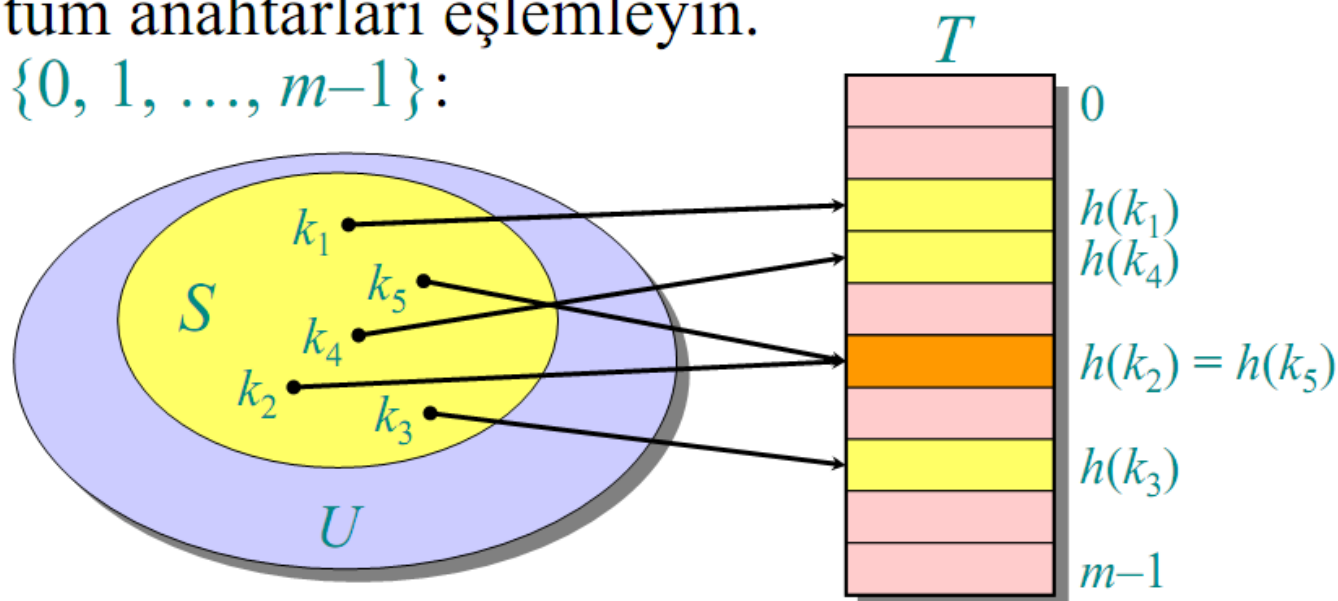
- Hashing, elimizdeki veriyi kullanarak o veriden elden geldiği kadar benzersiz bir tamsayı elde etme işlemidir.
- Bu elde edilen tamsayı, dizi şeklinde tutulan verilerin indisi gibi kullanılarak verilere tek seferde erişmemizi sağlar.



HASHING

(KIYIM FONKSİYONU)

Çözüm: *Kiyım fonksiyonu* h ile U evrenindeki tüm anahtarları eşlemleyin.
 $\{0, 1, \dots, m-1\}$:

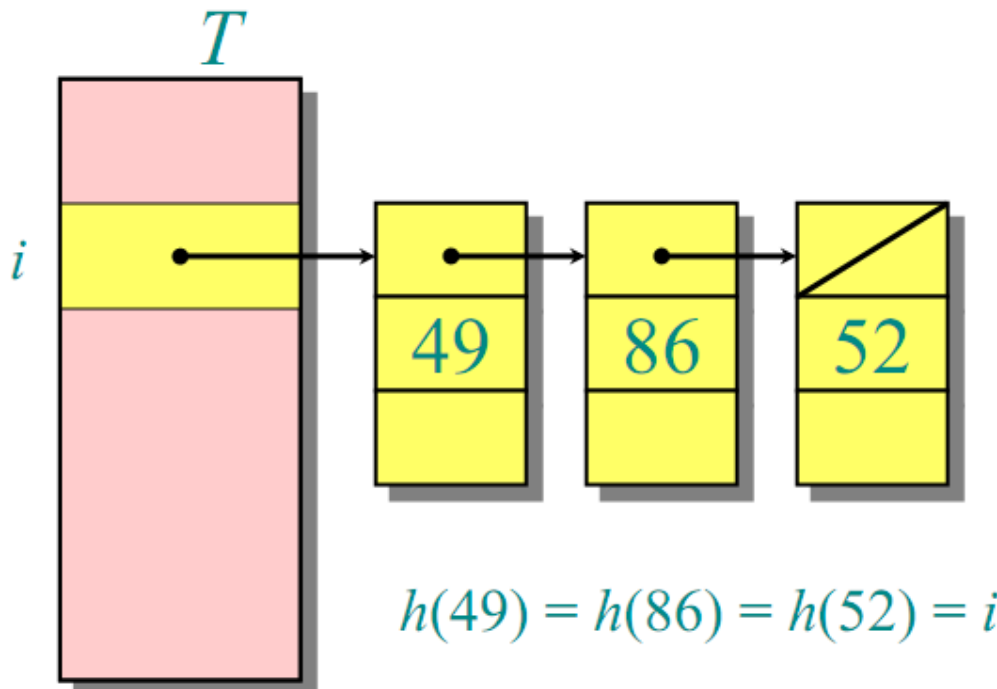


Araya yerleştirilecek kayıt T 'deki dolu bir yuvaya eşlemlendiğinde, bir **çarpışma** oluşur.

Çarpışmaları ilmeklemeyle (Chaining) çözme

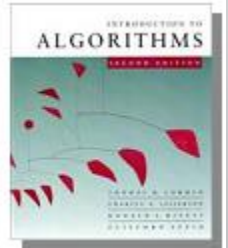


- Aynı yuvadaki kayıtları bir listeye ilişkilendirin.



En kötü durum:

- Tüm anahtarlar aynı yuvaya kısımlanır.
- Erişim süresi = $\Theta(n)$ eğer $|S| = n$ ise



İlmeklemede Ortalama Durum Çözümlemesi

Basit tekbiçimli kısımlama için şu varsayımı yaparız:

- Her anahtar $k \in S$, T tablosunun her yuvasına diğer anahtarların nereye kısımlandığından bağımsız olarak kısımlanır.

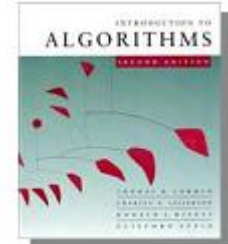
n bu tablodaki anahtarların sayısı ve m de yuvaların sayısı olsun.

T 'nin *yük oranını* tanımlarken;

$$\alpha = n/m$$

= yuva başına ortalama anahtar sayısıdır.

Arama maliyeti



Belirli bir anahtar kaydı için **başarısız** bir aramadaki beklenen süre

$$= \Theta(1 + \alpha).$$

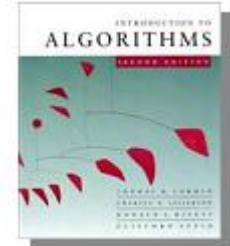
listeyi arama

kıyım fonksiyonu uygulama ve yuvaya erişim

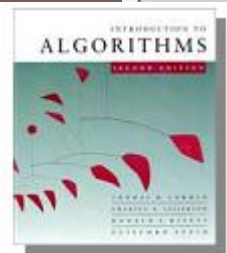
Beklenen arama süresi $= \Theta(1)$, eğer $\alpha = O(1)$, veya eğer $n = O(m)$ ise..

Başarılı bir arama da aynı asimptotik sınıra sahiptir, ama çok sıkı bir argüman biraz daha karmaşıktır.

Bir kıyım fonksiyonu seçmek



- Basit tek biçimli kıyımlamanın varsayımını garanti etmek zordur, ama eksikliklerinden kaçınılabildiği sürece pratikte iyi çalışan bazı ortak teknikler vardır.
- **İstenilenler:**
- İyi bir kıyım fonksiyonu, anahtarları tablonun yuvalarına tek biçimli dağıtabilmelidir.
- Anahtar dağılımındaki düzenlilik bu tek biçimliliği etkilememelidir.



Bölme metodu

Tüm anahtarların tam sayı olduğunu kabul edin ve şöyle tanımlayın: $h(k) = k \bmod m$ (ölçke) m .

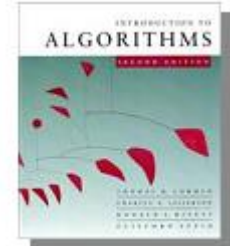
Sakınca: m' yi küçük bir d böleni olacak şekilde seçmeyin. Anahtarlardan çoğu ölçke (modulo) d ile çakışırsa, bu durum tekbiçimliliği olumsuz etkiler.

Uç sakınca: Eğer $m = 2^r$ ise, kıyım fonksiyonu k' nın bütün bitlerine bağımlı bile olmaz:

- Eğer $k = 1011000111\underbrace{011010}_2$ ve $r = 6$ ise, $m = 2^6$
 $h(k) = 011010_2$ dır. $h(k)$

Bölme metodu

$$h(k) = k \bmod m.$$



m 'yi , 2 veya 10' un bir kuvveti olmayacak şekilde ve bilgisayar dünyasında yaygın kullanılmayan asal sayılar arasından seçin.

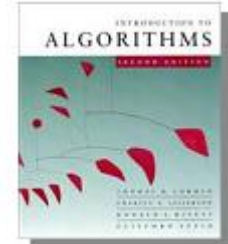
Rahatsızlık:

- Bazen, tablo boyutunu asal oluşturmak uygun değildir.

Buna rağmen bu metot yaygındır ama bir sonraki metot daha üstündür.

Not: m çift ve değerlerde çift sayı ise anahtarların hepsi aynı yuvayı işaret eder. Tek sayılı yuvalara hiçbir zaman kırım olmaz. Yuvaların yarısı boş olur. m yi asal seçmek daha uygundur ama her zaman değil, asal sayı 2 ve 10 nun kuvvetlerine yakın olmazsa iyidir.

Çarpma metodu

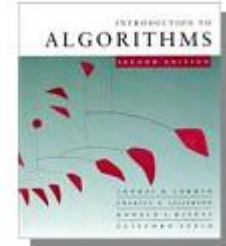


Tüm anahtarların tamsayı $m = 2^r$, ve bilgisayar sözcüklerinin de w -bit olduğunu kabul edin.

$h(k) = (A \cdot k \bmod 2^w) \text{ rsh } (w - r)$ 'yi tanımlayın, burada rsh “bit bazında sağa kayma” işlemcisi ve A da $2^{w-1} < A < 2^w$ aralığında tek tamsayı olsun.

- A 'yı 2^{w-1} veya 2^w 'ye çok yakın seçmeyin.
- Ölçke (modulo) 2^w ile çarpma bölmeye oranla daha hızlıdır.
- rsh (bit bazında sağa kaydırma) operatörü hızlıdır.

Çarpma metodu örneği



$$h(k) = (A \cdot k \bmod 2^w) \text{ rsh } (w - r)$$

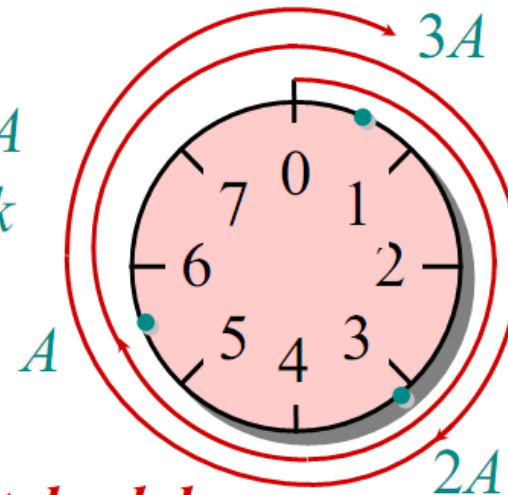
$m = 8 = 2^3$ olsun ve bilgisayarımızda da $w = 7$ -bit sözcükler olsun:

$$\begin{array}{r} \text{89} \quad 1\ 0\ 1\ 1\ 0\ 0\ 1 = A \\ \times \quad 107 \quad 1\ 1\ 0\ 1\ 0\ 1\ 1 = k \\ \hline 9523 \quad 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1 \end{array}$$

$h(k)$

mod 2^w alınırsa bu kısım ihmal edilir. Düşük değerli bitler kalır.

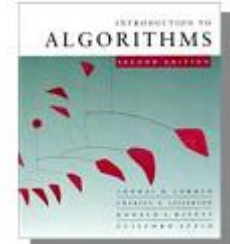
Modüler tekerlek



Burada A kesirli sayı düşünüldü ($A=11/2$)

Eğer A, örneğin tek sayı ise ve ikinin kuvvetlerinden birine çok yakın değilse, atamayı başka bir yerdeki farklı yuvaya yapar. Böylece etrafta dolaşırken k çok büyük bir değerse, k çarpı A çevrede k kere döner.

Çarpma metodu örneği



Multiplication Method

- Hash function is defined by size plus a parameter A
$$h_A(k) = \lfloor \text{size} * (k * A \bmod 1) \rfloor \text{ where } 0 < A < 1$$

- Example: size = 10, $A = 0.485$

$$\begin{aligned} h_A(50) &= \lfloor 10 * (50 * 0.485 \bmod 1) \rfloor \\ &= \lfloor 10 * (24.25 \bmod 1) \rfloor = \lfloor 10 * 0.25 \rfloor = 2 \end{aligned}$$

Açık adresleme ile çarpışmaları çözmek



Kıyım tablosunun dışında depo alanı kullanılmaz.

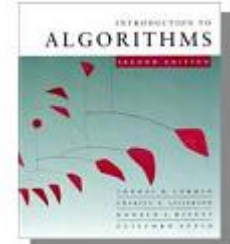
- Araya yerleştirme boş bir yuva bulunana kadar tabloyu sistematik biçimde sondalar.
- Kıyım fonksiyonu hem anahtara hem de sonda sayısına bağlıdır:

$$h : U \times \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}.$$

- Sonda dizisi $\langle h(k,0), h(k,1), \dots, h(k,m-1) \rangle$ $\{0, 1, \dots, m-1\}$ ' in bir permütasyonu olmalıdır.
- Tablo dolabilir ve silme işlemi zordur, (ama imkansız değildir).

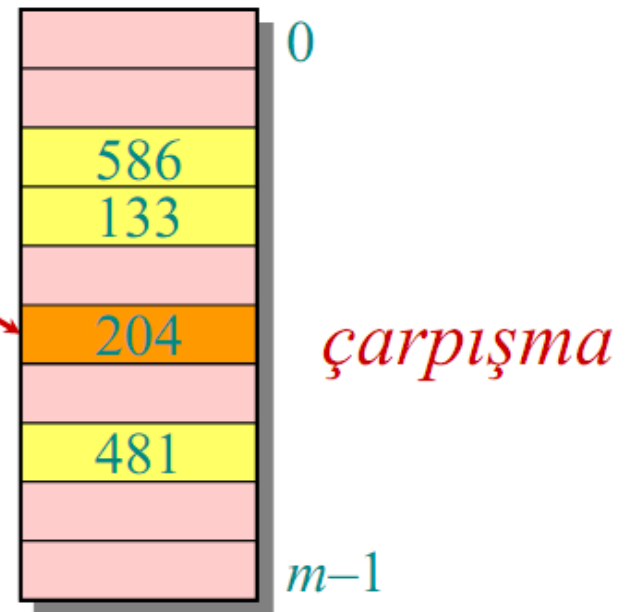
Tablo dolabilir olduğundan $n \leq m$ olmalıdır. Tablo dolarsa her yerde arama yapmak zorunda kalırız ve aradığımız elemanı bulamayabiliriz.

Açık adresleme için örnek

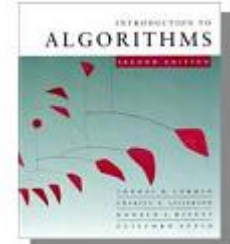


Anahtarı $k = 496$ araya yerleştirin:

0. Sonda $h(496, 0)$



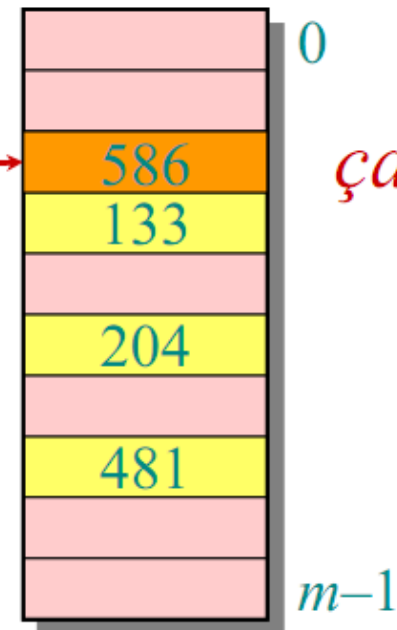
Açık adresleme için örnek



Anahtarı $k = 496$ araya yerleştirin:

0. Sonda $h(496, 0)$

1. Sonda $h(496, 1)$ →



Açık adresleme için örnek

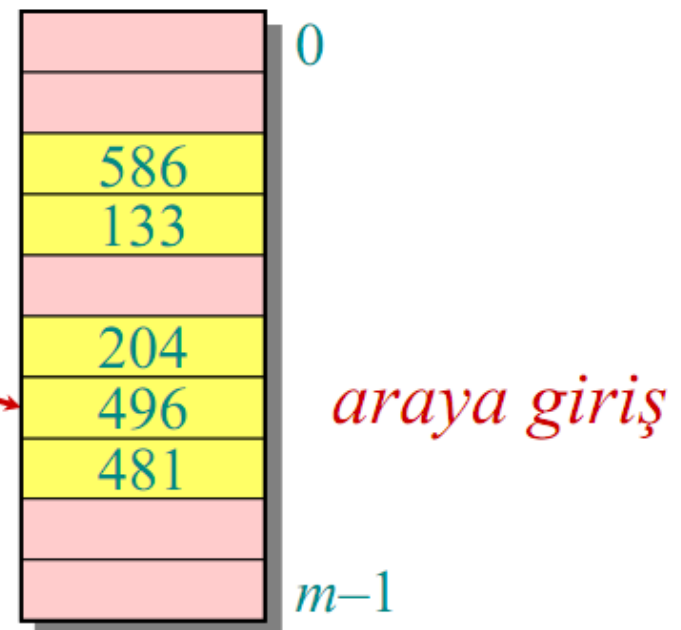


Anahtarı $k = 496$ araya yerleştirin:

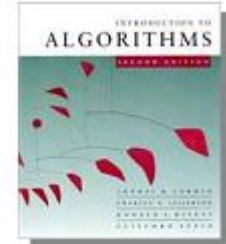
0. Sonda $h(496,0)$

1. Sonda $h(496,1)$

2. Sonda $h(496,2)$



Açık adresleme için örnek



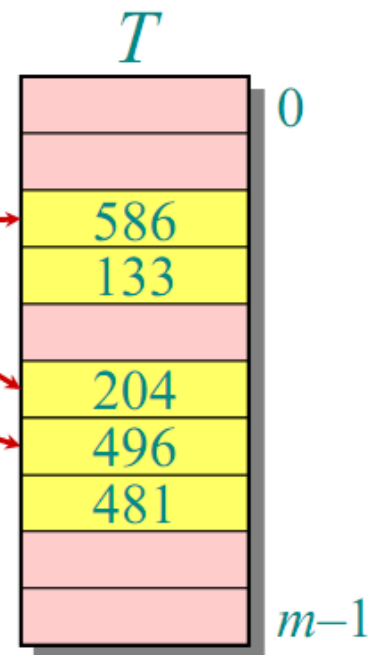
$k = 496$: Anahtarı arama

0. Sonda $h(496,0)$

1. Sonda $h(496,1)$

2. Sonda $h(496,2)$

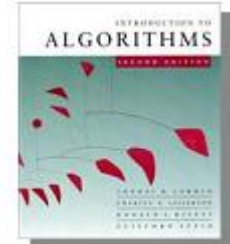
Arama da aynı sonda dizisini kullanır; anahtarı bulursa başarıyla, boş bir yuvayla karşılaşırsa başarısızlıkla sona erer.



Sondalama (Probing) Stratejileri

- Doğrusal Sondalama (Linear Probing)
 - – $h(k,i) = (h'(k) + i) \bmod m \rightarrow h(k,0)$
- İkinci Dereceden Sondalama(Quadratic probing)
 - – $h(k,i) = (h'(k) + c_1i + c_2i^2) \bmod m$
- Çift Kiyım (Double hashing)
 - – $h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod m$

Sonda stratejileri



Doğrusal sondalama:

$h'(k)$, gibi basit bir kıyım fonksiyonu verildiğinde, doğrusal sondalama şu kıyım fonksiyonunu kullanır:

$$h(k,i) = (h'(k) + i) \bmod m.$$

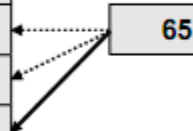
Bu metot basit olmakla birlikte **asal gruplandırma** sıkıntı yaratır; dolu yuvalar uzun sıralar oluşturur ve ortalama arama süresi artar. Ayrıca, dolu yuvaların sıra uzunluğu giderek artar.

Hash fonksiyonları

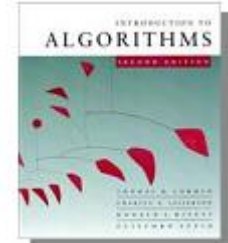
Çakışmanın giderilmesi (Linear Probing)

- Aynı pozisyona gelen ikinci kayıt ilgili pozisyondan sonraki ilk boş pozisyona yerleştirilir.
- Ekleme: Boş bir alan bulunarak yapılır.
- Silme/Erişim: İlk boş alan bulunana kadar devam edebilir.

0	-
1	-
2	47
3	-
4	-
5	35
6	36
7	65
8	-
9	129
10	25
11	2501
12	-
13	-
14	-



Sonda stratejileri



Linear Probing: Example

0	
1	
2	
3	3
4	13
5	5
6	6
7	23
8	15
9	

$$h(k,i) = (h'(k) + i) \bmod m$$

Ex: $m = 10$

Input = $\langle 5, 3, 6, 13, 23, 15 \rangle$

$$h(3,0) = (h'(3) + 0) \bmod 10 = 3$$

$$h(13,0) = (h'(13) + 0) \bmod 10 = 3 ?$$

$$h(13,1) = (h'(13) + 1) \bmod 10 = 4$$

$$h(5,0) = (h'(5) + 0) \bmod 10 = 5$$

$$h(6,0) = (h'(6) + 0) \bmod 10 = 6$$

$$h(23,0) = (h'(23) + 0) \bmod 10 = 3 ?$$

$$h(23,1) = (h'(23) + 1) \bmod 10 = 4 ?$$

$$h(23,4) = (h'(23) + 4) \bmod 10 = 7$$

Hash fonksiyonları

Çakışmanın giderilmesi (Linear Probing)

- Linear Probing metodunun avantajları / dezavantajları
- Bağlı listeler gibi ayrı bir veri yapısına ihtiyaç duyulmaz.
- Kayıtların yığın şeklinde toplanmasına sebep olur.
- Silme ve arama işlemleri için gereken zaman aynı hash değeri sayısı arttıkça artar.

Hash fonksiyonları

Çakışmanın giderilmesi (Quadratic Probing)

- Aynı pozisyona gelen ikinci kayıt Quadratic Fonksiyonla yerleştirilir.
- En çok kullanılan hash fonksiyonu
- $h(k,i) = (h'(k) + c_1i + c_2i^2) \bmod m$
- Burada h' , yardımcı hash fonksiyonu, c_1 ve $c_2 \neq 0$
- ve $i = 0, 1, \dots, M-1$.
- Sondalamanın başlangıç pozisyonu: $t = [h'(k)]$
- $h(k,i) = (t + c_1i + c_2i^2) \bmod m$

Quadratic Probing: Example

0	12
1	
2	2
3	3
4	
5	22
6	
7	
8	18
9	

$$h(k,i) = (h'(k) + c_1i + c_2i^2) \bmod M$$

Ex: $M = 10, c_1 = 2, c_2 = 1$

Input = $\langle 2, 3, 22, 12, 18 \rangle$

$$h(2,0) = (h'(2) + 2*0 + 1*0) \bmod 10 = 2$$

$$h(3,0) = (h'(3) + 2*0 + 1*0) \bmod 10 = 3$$

$$h(22,0) = (h'(22) + 2*0 + 1*0) \bmod 10 = 2 ?$$

$$h(22,1) = (h'(22) + 2*1 + 1*1) \bmod 10 = 5$$

$$h(12,0) = (h'(12) + 2*0 + 1*0) \bmod 10 = 2 ?$$

$$h(12,1) = (h'(12) + 2*1 + 1*1) \bmod 10 = 5 ?$$

$$h(12,2) = (h'(12) + 2*2 + 1*4) \bmod 10 = 0$$

$$h(18,0) = (h'(18) + 2*0 + 1*0) \bmod 10 = 8$$

Hash fonksiyonları

Çakışmanın giderilmesi (Quadratic Probing)

- Quadratic Probing metodunun avantajları / dezavantajları
- Anahtar değerlerini linear probing metoduna göre daha düzgün dağıtır.
- Yeni eleman eklemede tablo boyutuna dikkat edilmezse sonsuza kadar çalışma riski vardır.

Sonda stratejileri



Çifte kıyımlama

$h_1(k)$ ve $h_2(k)$, gibi iki basit kıyım fonksiyonu varsa, çifte kıyımlama şu kıyım fonksiyonunu kullanır:

$$h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod m.$$

Bu metot genelde mükemmel sonuçlar verir, ama $h_2(k)$, m 'e göre asal olmalıdır. Bunun bir yolu m 'yi, 2 'nin bir kuvveti yapmak ve $h_2(k)$ 'yı sadece tek sayılar üretecek şekilde tasarlamaktır.

Double Hashing

0	
1	79
2	
3	
4	
5	98
6	
7	
8	
9	14
10	
11	
12	

$$h(k,i) = (h_1(k) + i * h_2(k)) \bmod M$$

Ya da

- $M=2^d$ ve h_2 çift sayı üretecek şekilde tasarlanabilir
- M asaldır ve h_2 , M 'den daha küçük pozitif tam sayı üretecek şekilde tasarlanır.

Ex: $M = 13$, $M' = 11$ $\leftarrow M'$ should be slightly less than M

$\rightarrow h_1(k) = k \bmod M$, $h_2(k) = 1 + (k \bmod M')$.

Input = <98, 79, 14>

$$h_1(98) = 98 \bmod 13 = 5$$

$$h_1(79) = 79 \bmod 13 = 1$$

$$h_1(14) = 14 \bmod 13 = 1$$

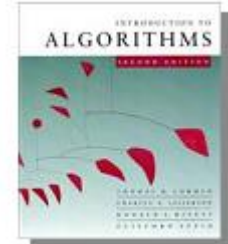
$$h_2(14) = 1 + 14 \bmod 11 = 4$$

$$h(14, 1) = (h_1(14) + 1 * h_2(14)) \bmod 13 = 1 + 1 * 4 = 5$$

$$h(14, 2) = (1 + 2 * 4) \bmod 13 = 9$$

Çifte Kısımlama

Teoremin kanıtlanması

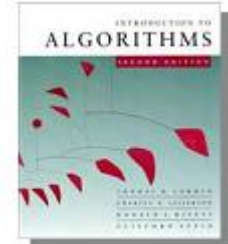


Kanıt.

- En az bir sondalama mutlaka gereklidir.
- n/m olasılığıyla, ilk sonda dolu bir yuvaya gider ve ikinci sondalama gerekli olur.
- $(n-1)/(m-1)$ olasılığıyla, ikinci sonda dolu bir yuvaya gider ve üçüncü sondalama gerekli olur.
- $(n-2)/(m-2)$ olasılığıyla, üçüncü sonda da dolu bir yuvaya gider, v.b.

Gözlemle: $\frac{n-i}{m-i} < \frac{n}{m} = \alpha$; $i = 1, 2, \dots, n$ için...

Teoremin kanıtlanması



Bu nedenle, beklenen sonda sayısı:

$$\begin{aligned}
 & 1 + \frac{n}{m} \left(1 + \frac{n-1}{m-1} \left(1 + \frac{n-2}{m-2} \left(\dots \left(1 + \frac{1}{m-n+1} \right) \dots \right) \right) \right) \\
 & \leq 1 + \alpha (1 + \alpha (1 + \alpha (\dots (1 + \alpha) \dots))) \\
 & \leq 1 + \alpha + \alpha^2 + \alpha^3 + \dots \\
 & = \sum_{i=0}^{\infty} \alpha^i \\
 & = \frac{1}{1-\alpha} \cdot \square
 \end{aligned}$$

Başlangıçta 1 sondalama olacaktır.

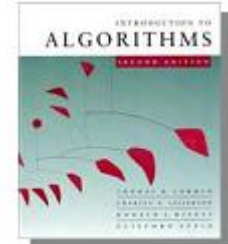
n/m çarpışma olacaktır.

2.sondada çarpışma olasılığı $(n-1)/(m-1)$ olacaktır. Böyle devam eder....

Geometrik Seriler: $\sum_{i=0}^N A^i = \frac{A^{N+1} - 1}{A - 1} \quad A > 1$

$\sum_{i=0}^N A^i = \frac{1 - A^{N+1}}{1 - A} = \Theta(1) \quad |A| < 1$

Teoremin açılımları

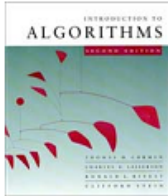


- Eğer α bir sabitse, açık adresli bir kısıym tablosuna erişim sabit zaman alır.
- Eğer tablo yarı doluysa, beklenen sonda sayısı $1/(1-0.5) = 2$ ' dir.
- Eğer tablo % 90 doluysa, beklenen sonda sayısı $1/(1-0.9) = 10$ ' dur.

Hash fonksiyonları

Çakışmanın giderilmesi (Double Hashing)

- Double Hashing metodunun avantajları / dezavantajları
- Çok iyi bir kıyım fonksiyonudur
- Anahtar değerlerini linear probing metoduna göre daha düzgün dağıtır ve gruplar oluşmaz.
- Quadratic probing metoduna göre daha yavaştır çünkü ikinci bir hash fonksiyonu hesaplanır.



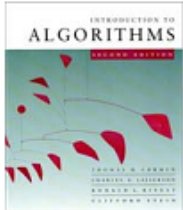
Kıyım fonksiyonunun bir zaafı

Problem: Her kıyım fonksiyonu h için, kıyım tablosuna ortalama erişim süresini çok büyük ölçüde arttıracak bir anahtar kümesi vardır.

- Rakibiniz bir i yuvası için tüm anahtarları $\{k \in U : h(k) = i\}$ 'den elde edebilir.

Fikir: Kıyım fonksiyonunu tüm anahtarlardan bağımsız olacak şekilde rastgele seçin.

- Rakibiniz kodunuzu görüyor olsa bile, hangi kıyım fonksiyonunun seçileceğini kesinlikle bilmediğinden, kötü bir anahtar kümesi bulamayacaktır.



Bir evrensel kıyım fonksiyonları setini yapılandırmak

m asal sayı olsun. k anahtarını $r + 1$ basamağa ayırıştırın; herbirinin set içinde değeri $\{0, 1, \dots, m-1\}$ olsun. Yani, $k = \langle k_0, k_1, \dots, k_r \rangle$ ve $0 \leq k_i < m$ olsun.

Rastgele yapma stratejisi:

$a = \langle a_0, a_1, \dots, a_r \rangle$ olsun; burada a_i , $\{0, 1, \dots, m-1\}$ arasından rastgele seçilmiştir.

Tanım: $h_a(k) = \sum_{i=0}^r a_i k_i \bmod m$. *Nokta çarpım, mod m (ölçke)*

Evrensel Kiyım

Good Hashing: Universal Hash Function

-
-
- Parameterized by prime size and vector:
 - $a = \langle a_0 \ a_1 \ \dots \ a_r \rangle$ where $0 \leq a_i < \text{size}$
- • Represent each key as $r + 1$ integers where $k_i < \text{size}$
 - size = 11, key = 39752 $\implies \langle 3, 9, 7, 5, 2 \rangle$
 - size = 29, key = “hello world” $\implies \langle 8, 5, 12, 12, 15, 23, 15, 18, 12, 4 \rangle$

$$h_a(k) = \left(\sum_{i=0}^r a_i k_i \right) \bmod \text{size}$$

Universal Hash Function: Example

Context: hash strings of length 3 in a table of size 131

let $a = \langle 35, 100, 21 \rangle$

$$\begin{aligned} h_a(\text{"xyz"}) &= (35 * 120 + 100 * 121 + 21 * 122) \% 131 \\ &= 129 \end{aligned}$$

Dezavantajı: k_i değeri tablo boyutundan büyük olabilir.

Bu yüzden tablo boyutunu k_i değerinden büyük seçilmeli

Mükemmel Kırım

- Şu ana kadar yaptıklarımız beklenen zamanda başarımla ilgiliydi. Kırım, beklenen süre bağlamında iyi bir uygulama. **Mükemmel kırım** ise şu sorulara ilgilenir: Farz edin ki size bir anahtar kümesi verildi ve bana statik bir tablo oluşturmanız istendi. Böylece en kötü zamanda tabloda anahtarı arayabileyim.
- Bir iyi birde en kötü zamanda. Dolayısıyla elimde sabit bir anahtar kümesi var. Aynı İngilizcedeki en sık kullanılan 100 veya 1000 sözcük gibi bir şey.

Mükemmel Kıyım

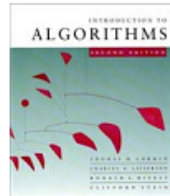
- Bir sözcük ele alındığında, sözcüğün İngilizcede sık kullanılıp kullanılmadığına tabloya bakarak hızlı bir şekilde anlamalıyız. Bu işi beklenen başarımla değil de garantilenmiş en kötü durum zamanında yapabilmeliyiz.
- Problem şu; verilen n adet anahtar için statik bir kıyım tablosu yaratmak. Diğer bir deyişle, yeni girdi veya silme yapılmayacak. Sadece elemanları oraya koyacağız. Büyüklüğü ise, $m = O(n)$.
- $m = O(n)$ boyutunda bir tablo ve en kötü durumda arama $O(1)$ zamanı alacak. Ortalama durumu biliyor olacağız, bu çok zor değil, ama en kötü durumda değerlerin yığılıp, fazla zaman kaybına neden olacağı bir nokta olmayacağından emin olmalıyız. Herhangi bir noktada bu olmamalı; her bir arama $O(1)$ zamanında olmalı.

Mükemmel Kıyım

- Buradaki fikir iki aşamalı bir veri tanımlaması yapmaktır. Fikir, kıyım yapmak; bir kıyım tablomuz olacak, yuvalara kıyım yapacağız, ancak zincirleme işlemi kullanmak yerine ikinci bir kıyım tablosu daha olacak. İkinci tabloya ikinci bir kıyım daha yapacağız. Ve buradaki fikir ikinci düzeyde hiç çarpışma olmadan kıyım yapmak.
- Dolayısıyla birinci düzeyde çarpışma olabilir. Birinci tabloda çarpışan her şeyi ikinci düzeydeki tabloya koyacağız, ama bu tabloda çarpışma olmayacak.
- Dolayısıyla evrensel bir kıyım fonksiyonu bulalım. Rastgele bir fonksiyon seçiyoruz. Yapacağımız bu düzeye yani ilk düzeye kıyım yapmak.

Mükemmel Kıyım

- Bundan sonra iki şeyi takip edeceğiz. Birincisi, diğer düzeydeki kıyım tablomuzun büyüklüğü. Bu durumda, kıyım tablomuzun büyüklüğünü yuva sayısıyla adlandıracağız. Örneğin 1. düzeyde kıyım fonksiyonu 1. yuvaya sondalansın ve değeri 4 olsun. İkinci düzey içinse farklı bir kıyım anahtarı kullanacağız.
- Dolayısıyla, ikinci düzeyde her yuvanın farklı bir kıyım fonksiyonu olacak. Mesela, bir yuva rastgele seçilmiş 31 değerini taşıyabilir. Sonra, kıyım tablosuna bir işaretçi koyayım; buna büyük S1 diyeyim. Bu 4 yuvaya sahip olacak ve 14 ile 27'yi saklayacak. Bu
- $h(14) = h(27)$ o da 1'e eşit. Çünkü birinci yuvadayız. Şimdi bu ikisi birinci düzeyde kıyım tablosunda aynı yuvaya kıyılıyor. Bu birinci düzeyde..

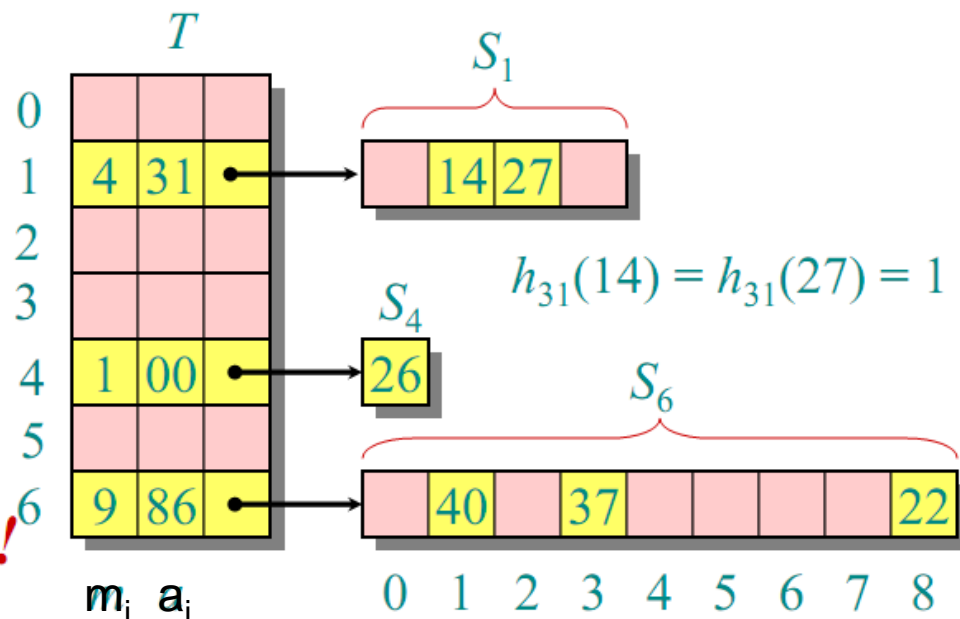


Mükemmel kıyım fonksiyonu

n anahtarlı bir set verilirse, bir statik kıyım tablosunu boyutu $m = O(n)$ olacak şekilde yapılandırın ve ARAMA (SEARCH) *en kötü durumda* $\Theta(1)$ süre alsın.

FİKİR: Her iki düzeyde de evrensel kıyım ile 2 düzeyli veri tanımlama.

2. düzey çarpışması yok!



Mükemmel Kıyım

- Buradaki de ikinci düzey. Yani 14 ve 27 birinci seviyede çarpıştılar ve aynı yuvaya gittiler. Ancak ikinci seviyede farklı yuvalara kıyıldılar. Seçtiğim kıyım fonksiyonu seçtiğim rastgele sayılara göre anahtar listesi oluşturarak bu yapıyı yarattı.
- İkinci düzeyde, $h_{31}(14)$ sayısı için, 1'e eşit ve $h_{31}(27)$ sayısı için 2 değerlerini aldı.
- Eğer kıyım tablosunun i. yuvasına kıyılan n_i tane elaman varsa, ikinci düzeydeki tabloda m_i sayıda yuva kullanırız ve burada m_i , n_i 'nin karesi kadar yuvaya eşit olarak seçilir.
- Örnek olarak, 2 elemanım varsa 4 büyüklüğünde bir kıyım tablom olur. 3 elemanım varsa 9 yuvalı bir kıyım tablosuna ihtiyacım olur.

Mükemmel Kiyım

- Örnek : $K=\{10,22,37,40,52,60,70,72,75\}$ 9 elamanlı bir anahtar kümesi mod yani $m=n=9$ olur. Hash fonksiyonumuz: $h(k)=((a*k+b) \bmod p) \bmod m$
- $a=3, b=42, p=101, m=9$ (a ve b değerleri 0-101 arasında rastgele üretilen sayılar) Öncelikle ilk kıyım tablomuzda çakışmaların sayısını bulalım
- $h(10)=0$ 0. ve 5. indiste 1 çakışma
- $h(60)=2$ 2. indiste 3 çakışma
- $h(72)=2$ 7. indiste 4 çakışma
- $h(75)=2$
- $h(70)=5$
- $h(22)=7$
- $h(37)=7$
- $h(40)=7$
- $h(52)=7$

	n_i
0	1
1	
2	3
3	
4	
5	1
6	
7	4
8	

Mükemmel Kiyım

- Çakışmaları bulduktan sonra tek çakışmaya sahip değerler için a_i ve b_i değerlerini 0, diğerleri için ise 0-p arasında random seçelim, 2.kiyım (S_i) tablosunun büyüklüğü ise $m_i = n_i^2$ olacak
- 2.kiyım fonksiyonunda çakışma olmayacak şekilde yapılandırılalım
- $h_i(k) = ((a_i * k + b_i) \bmod p) \bmod m_i$

	n_i	m_i	a_i	b_i
0	1	1	0	0
1				
2	3	9	10	18
3				
4				
5	1	1	0	0
6				
7	4	16	23	88
8				

10
0

S_0

			60	72			75	
0	1	2	3	4	5	6	7	8


S_2

70
0

S_5

			40					52	22					37	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

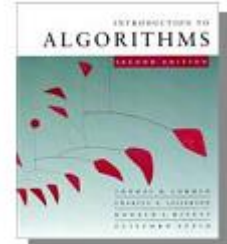
S_7



İkili Arama Ağaçları (BST)

Rastgele yapılanmış ikili
arama ağaçları

- Beklenen düğüm derinliği
- Yüksekliği çözümlmek



İkili-arama-ağacı sıralaması

$T \leftarrow \emptyset$ $i = 1$ den n' ye kadar değiştiğinde,

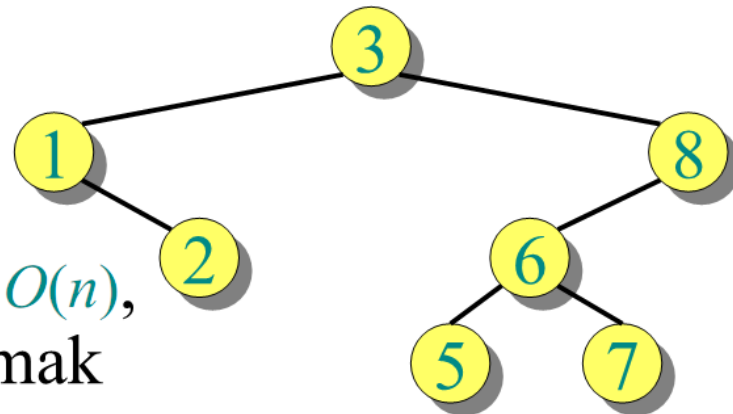
▷ Boş bir BST (ikili arama ağacı) yarat.

AĞAÇ ARAYA YERLEŞTİRMESİ **YAP** ($T, A[i]$)

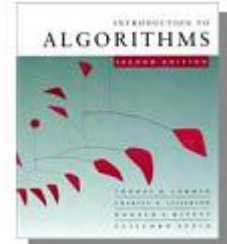
T 'nin içinde sıralı adımlama yap.

Örnek:

$A = [3 \ 1 \ 8 \ 2 \ 6 \ 7 \ 5]$

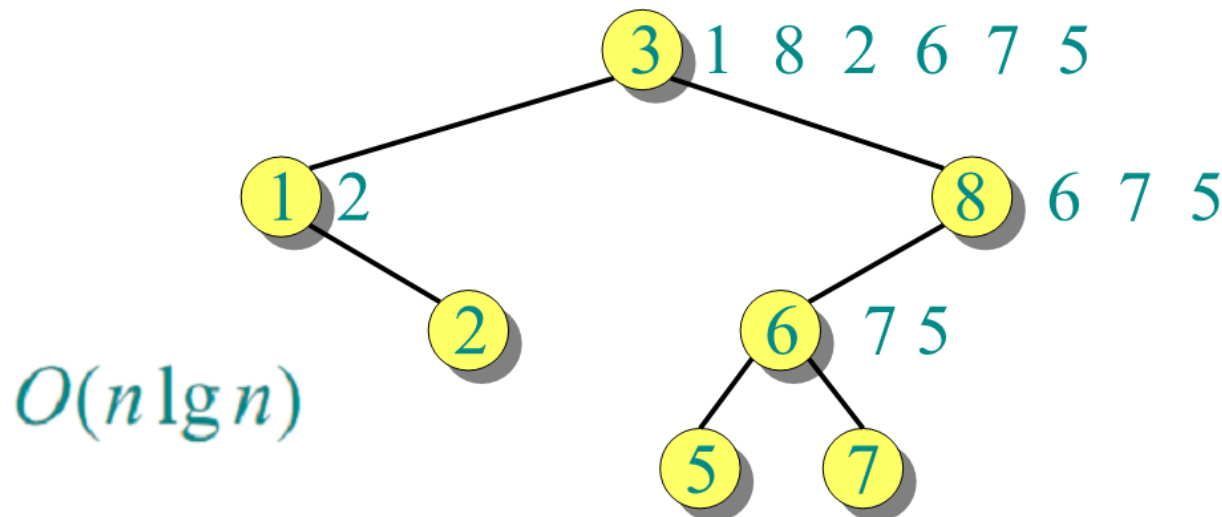


Ağaç adımlama süresi = $O(n)$,
ancak BST'yi oluşturmak
ne kadar zaman alır?

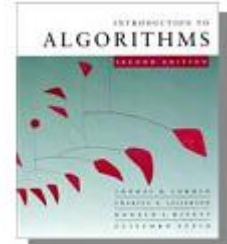


BST sıralaması çözümlemesi

BST sıralaması çabuk sıralama karşılaştırmalarının aynısını, başka bir düzende yapar!



Ağacı oluşturmanın beklenen süresi asimptotik olarak çabuk sıralamanın koşma süresinin aynıdır.



Düğüm derinliği

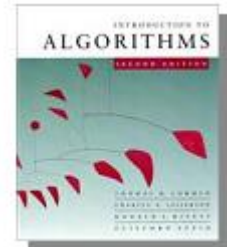
Bir düğüm derinliği = AĞAÇ ARAYA YERLEŞTİRMESİ için yapılan karşılaştırmalar. Tüm girdi permütasyonları eşit olasılıklı varsayılırsa:

Ortalama düğüm derinliği

$$= \frac{1}{n} E \left[\sum_{i=1}^n \text{(Boğum } i' \text{ yi araya yerleştirmek için gerekli karşılaştırmaların sayısı)} \right]$$

$$= \frac{1}{n} O(n \lg n) \quad (\text{Çabuk sıralama analizi})$$

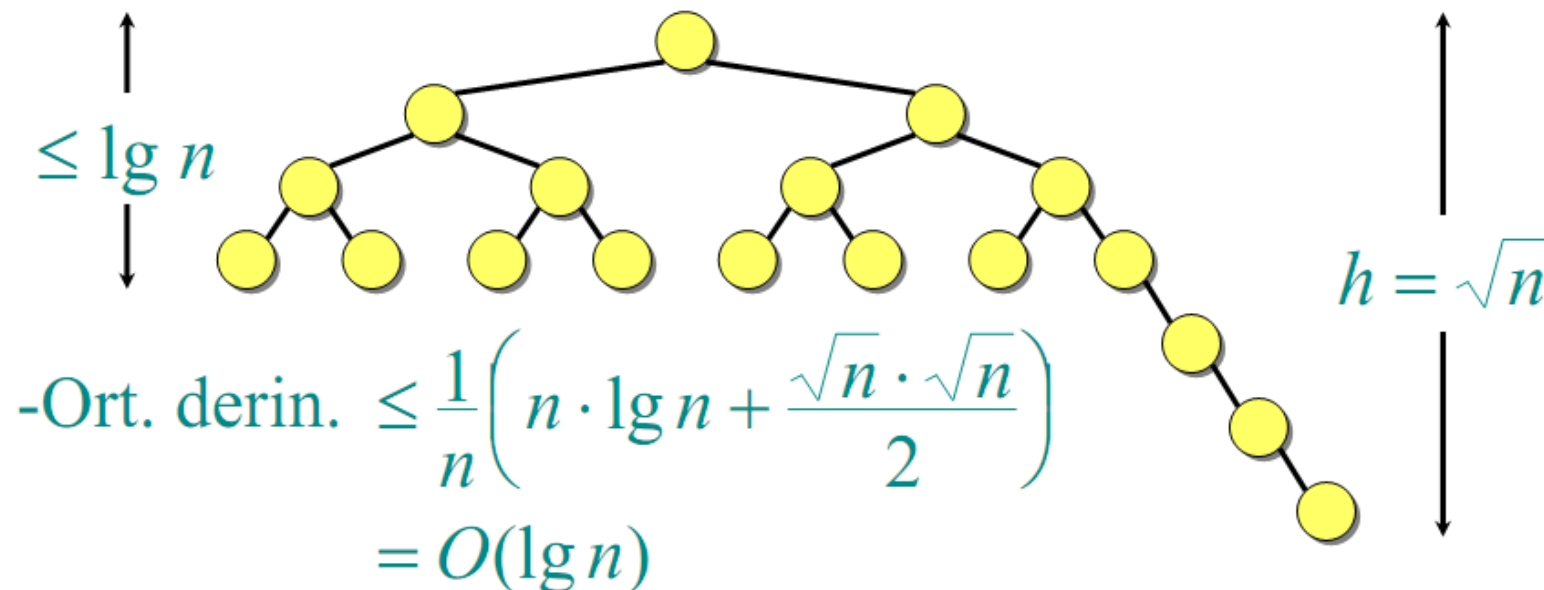
$$= O(\lg n) .$$



Ağacın beklenen yüksekliği

Ama, ortalama düğüm derinliğinin rastgele yapılanmış bir ikili arama ağacında (BST) $= O(\lg n)$ olması ağacın beklenen yüksekliğinin de $O(\lg n)$ olduğu anlamına gelmeyebilir (buna rağmen öyledir).

Örnek.



Dengeli arama ağaçları

Dengeli arama ağacı: n elemanlı bir değişken kümede işlem yaparken $O(\lg n)$ yüksekliğinin garanti edildiği bir arama ağacı veri yapısı.

Örnekler:

- AVL ağaçları
- 2-3 ağaçları
- 2-3-4 ağaçları
- B-ağaçları
- Kırmızı-siyah ağaçlar



7.Hafta

Dengeli Arama Ağaçları (Red - Black Tree)

- Kırmızı-siyah ağaçlar
 - Kırmızı-siyah ağacın yüksekliği
 - Rotation / Dönme
 - Insertion / araya yerleştirme
-