



ARAMA (SEARCHING)

# ARAMA (SEARCHING)



- ❖Günümüzde birçok arama algoritması geliştirilmiştir.
- ❖Arama işlemlerinin etkin bir şekilde yapmak için Sembol tablosundan yararlanılmaktadır.
- ❖**Sembol tablosu terimi**, *bilgiyi (bir değer) kaydedip bir anahtar kullanarak tekrar arama işlemi yapmak için kullanılan soyut bir mekanizmadır.*
- ❖Uygulamaya göre anahtar ve değer doğası değişmektedir.

# SEMBOL TABLOSU

- ❑ Sembol tablolarının en önemli amacı **bir değer ve anahtarı birbirine** bağlamaktır.
- ❑ Tabloya yerleştirilmiş olan anahtar- değer çiftleri arasında sonra arama yapmak için istemci anahtar değer çiftlerini tabloya girebilir.

## Tipik sembol tablosu uygulamaları

uygulama	arama sebebi	anahtar	değer
<i>sözlük</i>	anlamı bulmak	kelime	anlamı
<i>kitap indeksi</i>	ilgili sayfaları bulmak	terim	sayfa numarası listesi
<i>dosya paylaşımı</i>	indirmek için şarkı bulmak	şarkının ismi	bilgisayar ID'si
<i>hesap yönetimi</i>	etkileşimleri işlemek	hesap numarası	işlem detayı
<i>web araması</i>	ilgili web sayfasını bulmak	anahtar kelime	sayfa isimleri listesi
<i>derleyici</i>	tip ve değeri bulmak	değişken adı	tip ve değeri

# SEMBOL TABLOSU

❖ Çok fazla sayıda anahtara ve büyük miktarda veriye sahip olabiliriz.

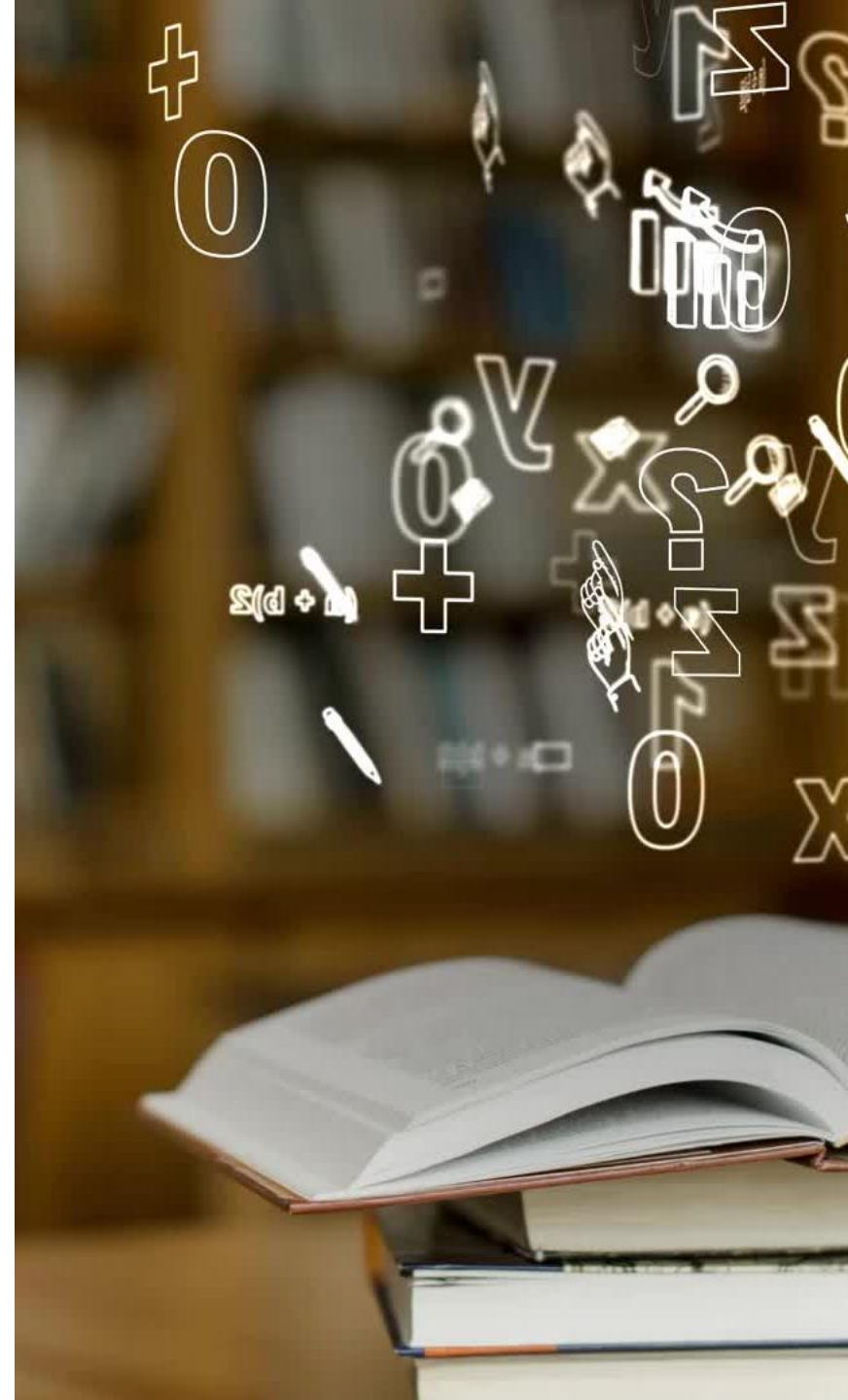
Dolayısıyla etkin bir sembol tablosu oluşturmak zordur.

❖ Sembol tabloların etkin bir şekilde gerçekleştirilmesi için 3 temel klasik veri yapısı aşağıda verilmiştir.

1. Hash (Kıyım, çırpı, karım) tabloları
2. İkili arama ağaçları
3. Kırmızı siyah ağaçlar

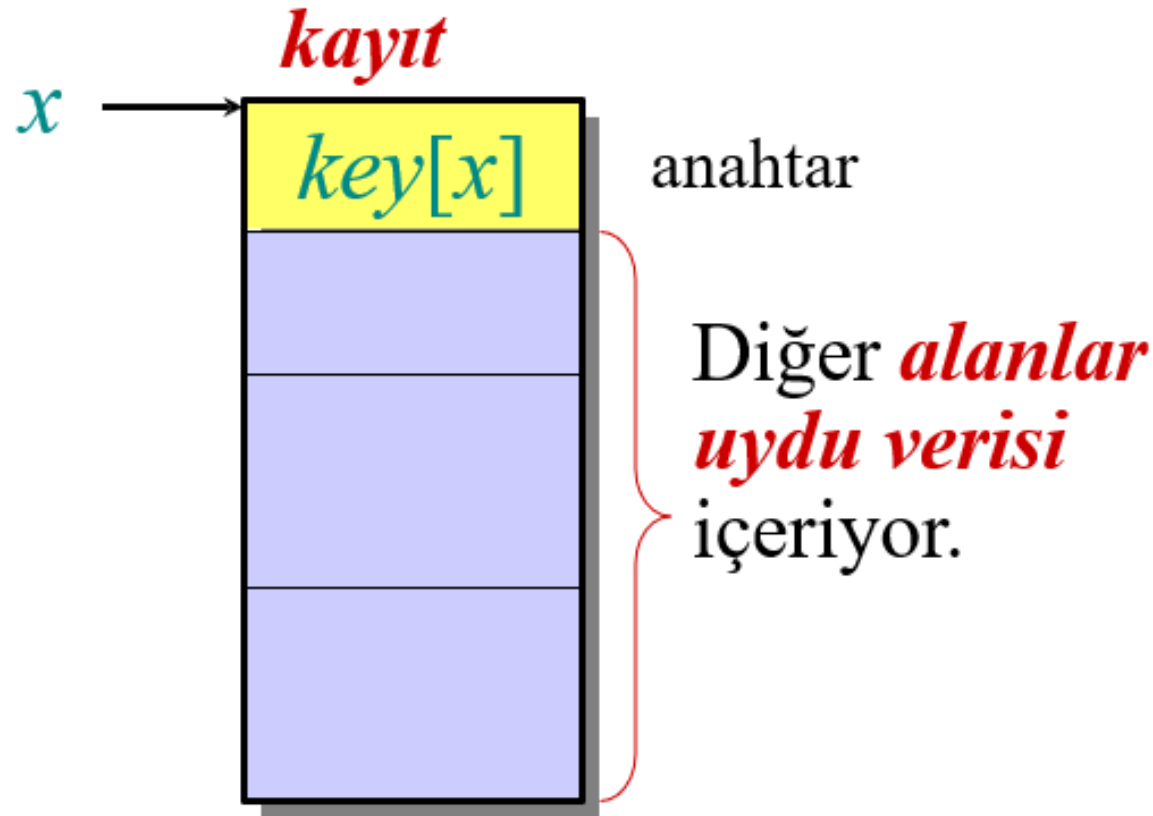
# Hash (Kıyım, Karım, Çırpı) Tablosu

- ❑ Kıyım (hash) tablosu, sözlük uygulamaları için etkili bir veri yapısıdır.
- ❑ Kıyım tablosunda bir **elemanı aramak**, bir bağlı liste içinde eleman aramak kadar uzun sürse de (pratikte en kötü ihtimalle)  $\Theta(n)$  **zamanda**, çırpılama çok iyi işlemektedir.
- ❑ Mantıklı varsayımlar üzerinde, kıyım tablosunda **ortalama bir eleman arama süresi  $O(1)$ 'dir.**



# Hash (Kıyım, Karım, Çırpı) Tablosu

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 Adres Tabloları

- ❑ Doğrudan adresleme, anahtar evreni mantıklı derecede küçük olduğunda iyi çalışan basit bir yöntemdir.
- ❑ Dolayısıyla bir uygulama her bir elemanın  *$m$ 'nin çok büyük olmadığı farz edilir*
- ❑  $U = \{0,1, \dots, m - 1\}$  evreninden çekilen bir anahtara sahip olduğu dinamik bir kümeye ihtiyaç duysun.

# Doğrudan Adres Tabloları

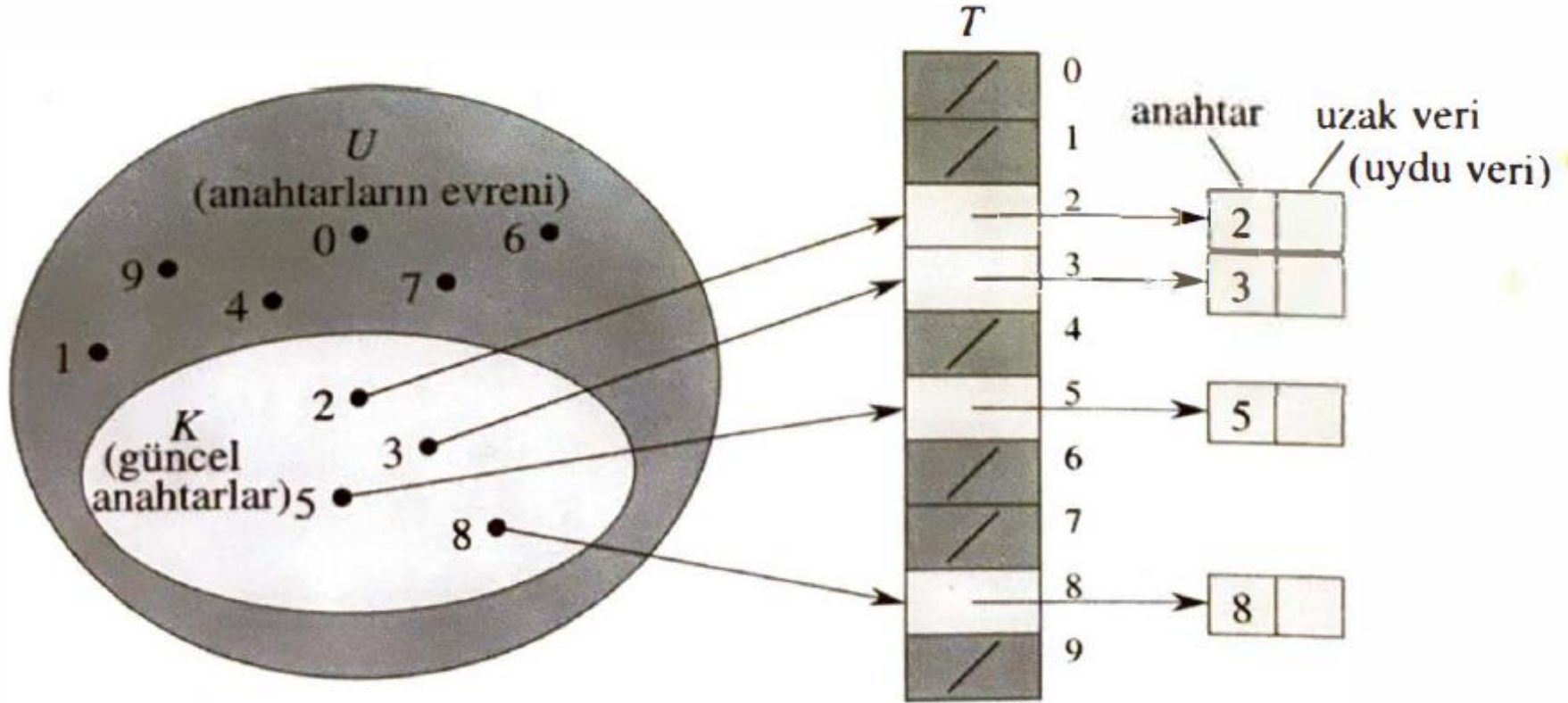
**FİKİR:** Anahtarların  $U \subseteq \{0, 1, \dots, m-1\}$  setinden seçildiğini ve birbirlerinden farklı olduklarını varsayın.  $T[0 \dots m-1]$  dizilimini oluşturun:

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

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



# Doğrudan Adres Tabloları



\*  $U = (0, 1, \dots, 9)$  evrenindeki her bir küme, **tablodaki bir indise denk gelir.**

\* Güncel anahtarların kümesi  $K = \{2, 3, 5, 8\}$ .

**Şekil 1.** Doğrudan-adres tablosu  $T$ , ile nasıl dinamik bir küme uygulandığını gösterir. Elemanların işaretçilerini içeren tablodaki pozisyonlarını belirler. Gölgelendirilmiş diğer pozisyonlar boştur.

# Doğrudan Adres Tabloları

DIRECT-ADDRESS-SEARCH ( $T, k$ )

1    **return**  $T[k]$

DIRECT-ADDRESS-INSERT ( $T, x$ )

1     $T[x, key] = x$

DIRECT-ADDRESS-DELETE ( $T, x$ )

1     $T[x, key] = \text{NIL}$

Her bir işlem  $O(1)$  zaman alır.

# Doğrudan Adres Tabloları

**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.

# Doğrudan Adres Tabloları

**Problem:** Bir başka problemde saklanan anahtar kümesi  $K$ ,  $U'$ 'ya göre oldukça küçük olabilir.

❑ Bu durumda doğrudan adresleme ile  $T$ , için ayrılan yerin çoğu boşa harcanmış olabilir.

❑ Bu olumsuz durumu ortadan kaldırmak için **kıyım(çırpı) tablosu** kullanılır.

❑ Kıyım tablosu daha az yere ihtiyaç duyar.

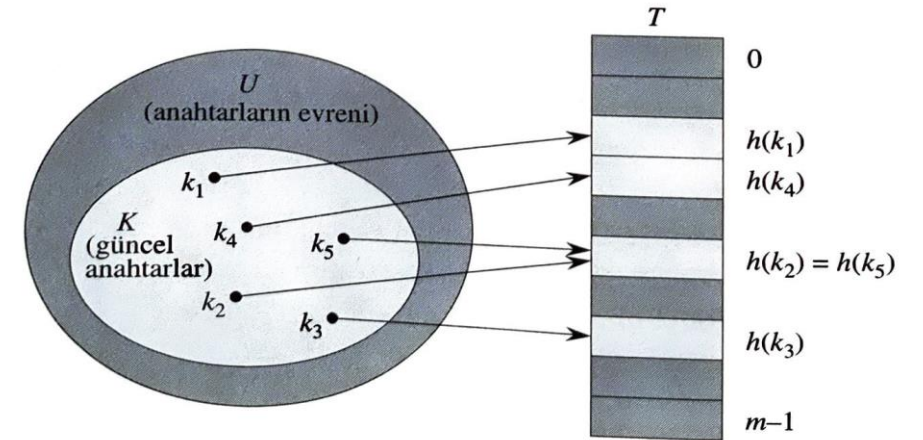
# Kıyımlama (Hashing)

Doğrudan adresleme ile  $k$  pozisyonunda  $k$  anahtarlı bir eleman saklanır.

Kıyımlama ile bu eleman  $h(k)$  pozisyonunda saklanır.

Yani,  $k$  anahtarından pozisyon hesaplamak için  $h$  kıyım fonksiyonunu kullanılır.

Burada  $h$ , kıyım *tablosu*  $T[0 .. m - 1]$  pozisyonlarının içindeki anahtarlar evreni  $U$ 'ya eşleme yapar.



# Kıyımlama (Hashing)

- Özet olarak kıyımlama (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.**

# Kıyımlama (Hashing)

---

Özellikle, kıyım tablosunun, **bir eleman aramak için  $O(1)$  zaman kullanır.**

---

Kıyım tablosunun **yer kaplama ihtiyacını  $\Theta(k)$  olarak düşünülebilir.**

---

**Bu sınır, ortalama-durum zamanı için geçerlidir.**

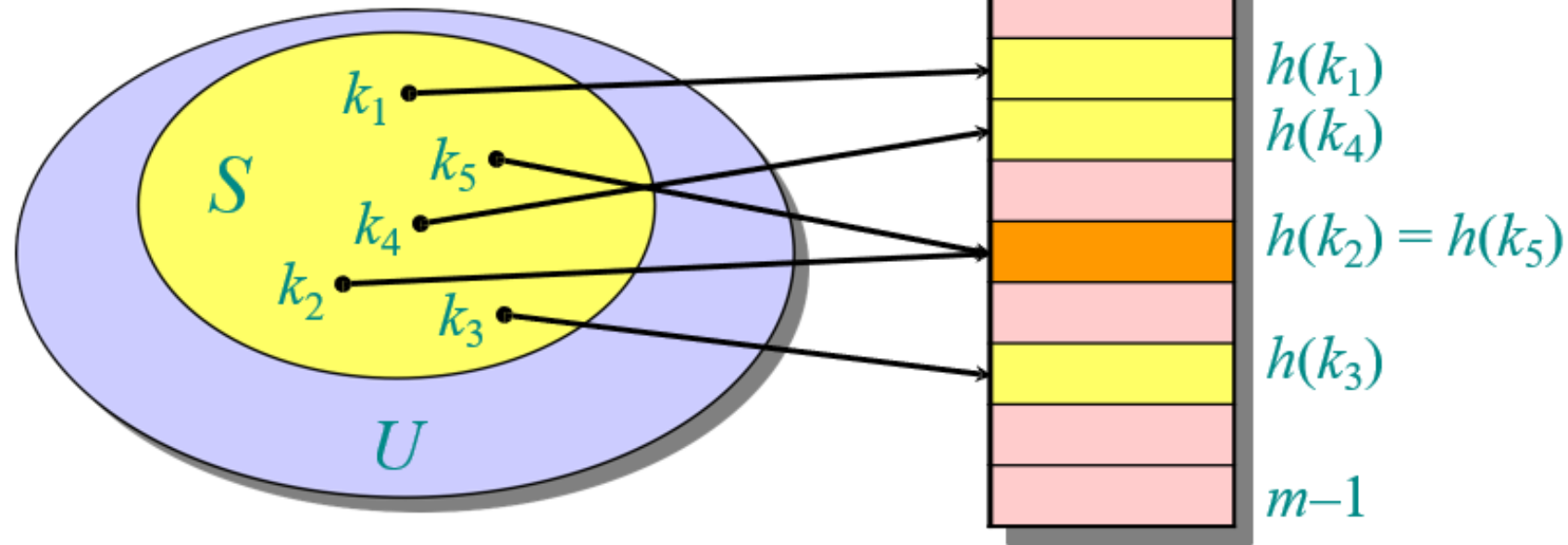
---

**Doğrudan adreslemede bu durum en kötü-durum zamanı için geçerlidir.**

# Kıyımlama (Hashing)

**Çözüm:** *Kıyım fonksiyonu*  $h$  ile  $U$  evrenindeki tüm anahtarları eşlemleyin.

$\{0, 1, \dots, m-1\}$ :

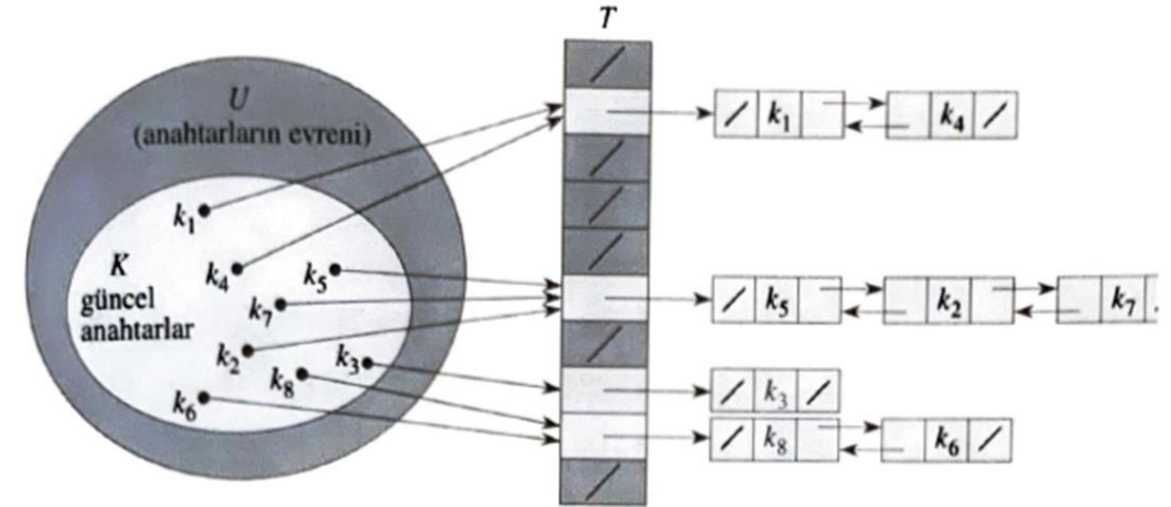


Araya yerleştirilecek kayıt  $T$ 'deki dolu bir yuvaya eşlemlendiğinde, bir *çarpışma* oluşur.



# Zincirleme ile çarpışma çözümü

- ❖ Her bir kıyım tablosu, pozisyonu  $T[j]$ , kıyım değeri  $j$  olan tüm anahtarların bağlı listesini içerir.
- ❖ Örneğin,  $h(k_1)) = h(k_4))$
- ❖ ve  $h(k_5) = h(k_7) = h(k_2)$
- ❖ Bağlı liste tekli veya çiftli olabilir.
- ❖ Yandaki şekilde çiftli liste olarak gösterilmiştir.
- ❖ Çünkü silme işlemi daha hızlıdır.



# Zincirleme ile kısımlama analizi

❖ Özellikle verilen anahtarlı bir elemanı aramak ne kadar sürer?

➤  $n$  tane eleman saklayan  $m$  tane pozisyonlu olarak verilen bir kiyım tablosu:

T için  $n/m$  olarak bir yükleme faktörü  $\alpha$  tanımlanır.

*Yani zincirde saklanan ortalama eleman sayısı*

$\alpha$  1'den küçük veya büyük ya da 1'e eşit olabilir.

# Zincirleme ile kısımlamanın en kötü durum davranışı

- *$n$  boyutlu bir liste* yaratılarak *tüm  $n$  anahtarları* aynı pozisyona/yuvaya kısımlar.
- Dolayısıyla en kötü durum zamanı  $\Theta(n)$  olur.
- Tüm elemanları saklamak için bağlı liste oluşturmaktan daha iyi değildir.

# Zincirleme ile kısımlamanın ortalama durum davranışı

- *Herhangi bir elemanın  $m$  pozisyonundan birine eşit şekilde kısımlandığını varsayalım.*
- *Bu varsayım **basit düzgün kısımlama** olarak adlandırılır.*

# Zincirleme ile kısımlamanın ortalama durum davranışı

***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.

*\* Bir başka ifadeyle herhangi bir elemanın  $m$  pozisyonundan birine eşit şekilde kısımlanması olarak kabul edilmesi*

# Arama maliyeti

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

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

*listeyi arama*

*kıyım fonksiyonu uygulama ve yuvaya erişim*

□ Bir başka ifadeyle tabloda önceden kayıtlı olmayan herhangi bir **k** anahtarı, m pozisyonundan herhangi birine kıyımlama eğiliminde olur.

□ Beklenen süre  $T[h(k)]$ , **listesinin sonunu aramak için beklenen süredir.**

□ Beklenen boyut  $E[n_{h(k)}] = \alpha$  dır.

Toplam gereken zaman  $\Theta(1 + \alpha)$  dır.

# Arama maliyeti

- ❑ **Başarılı** bir arama durumu biraz farklıdır.
- ❑ Çünkü her liste aramak için eşit benzerlikte değildir.
- ❑ Bunun yerine, *bir listenin aranma olasılığı*, onun içerdiği eleman sayısıyla doğru orantılıdır.
- ❑ Yine de, beklenen arama zamanı ortalama  $\Theta(1 + \alpha)$  civarındadır.

# 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.**



# Bir kırım fonksiyonu seçmek

❖ İyi bir kırımlama fonksiyonu tasarlamak için aşağıdaki üç temel yaklaşımlar kullanılır.

1. Bölme
2. Çarpma
3. Evrensel kırımlama

# 1. Bölme Metodu

□ Tüm anahtarların tamsayı olduğu kabul edilir.

Kıyım fonksiyonu:

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

Örneğin *kıyım tablosu*:  $m=12$  boyutundaysa;  
anahtar  $k=100$  ise  $h(k)=4$  olur.

# 1. Bölme Metodu

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

# 1. Bölme Metodu

**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,  
 $h(k) = 011010_2$  dır.  $h(k)$

- ❖ **k nın karakterlerinin sırasını değiştirmek onun kıyım değerini değiştirmez.**
- ❖ **m'nin 2'nin tam kuvvetlerine yakın olmayan bir asal sayı  $m$  için iyi bir seçim olabilir.**

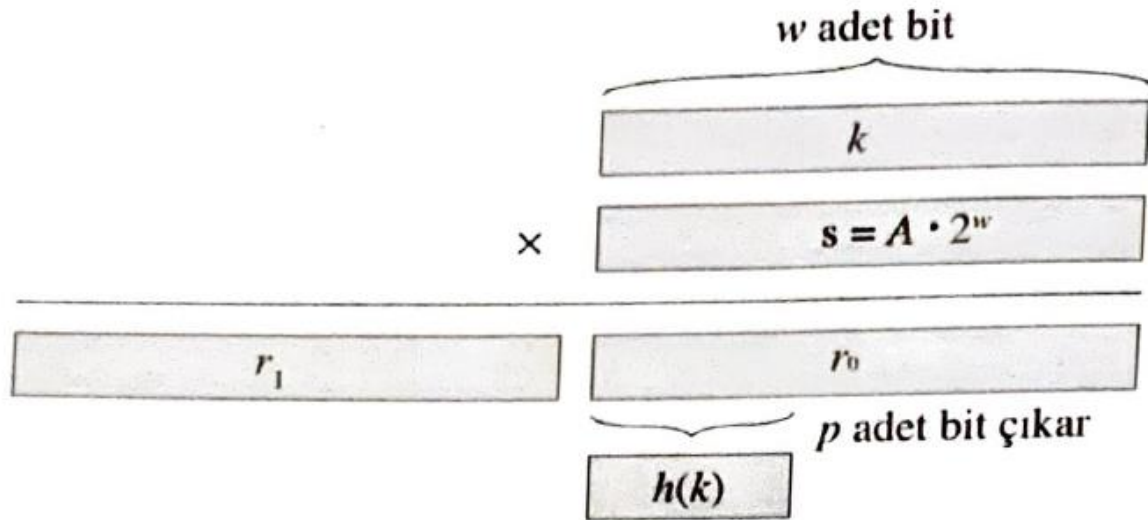
# 1. Bölme Metodu

- ❖  ***$m$  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.

# 1. Bölme Metodu

- ❖ Örneğin *karakterin 8 bit olduğu* kabaca  $n=2000$  karakter katarını tutmak için çarpışmaların zincirleme ile çözüldüğü bir kıyım tablosu oluşturmak istiyoruz.
- ❖ Başarısız bir aramada 3 elemanın ortalamasını incelemeyi önemsiyoruz.
- ❖ Bu yüzden  $m=701$  boyutunda bir çırpı tablosu oluştururuz.
- ❖ Çünkü  $m$ 'nin  $2000/3$  e yakın bir asal sayı ve  $2$ 'nin kuvvetine yakın değil.
- ❖ Her bir  $k$  anahtarına tam sayı gibi davranarak kıyım fonksiyonu
- ❖  $h(k)=k \bmod 701$

## 2. Çarpma Metodu



Şekil 2. Kısımlamanın çarpma yöntemi.

- ❑  $k$  anahtarının  $w$ -bit temsili,  $w$ -bit,  $s = (A \cdot 2^w)$  değeriyle çarpılır.
- ❑ Sonucun düşük  $w$ -bitinin bit yarısının  $p$  en yüksek-sıra bitleri, istenilen  $h(k)$  değerini oluşturur.

# Çarpma metodu

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

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

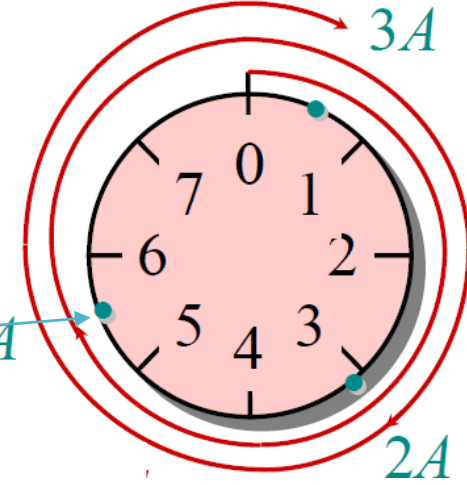
	89	1 0 1 1 0 0 1	= A
×	107	1 1 0 1 0 1 1	= k
<hr/>			
9523	1 0 0 1 0 1 0 0 1 1 0 0 1 1		

$h(k)$

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

Rsh : Sağa kaydır.  
Dolayısıyla yok olur.

## Modüler tekerlek



burada rsh “bit bazında sağa kayma” işlemcisi ve  $A$  da  $2^{w-1} < A < 2^w$  aralığında tek tamsayı olsun.

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

$$h_A(k) = \lfloor \text{size} * (k * A \bmod 1) \rfloor \text{ where } 0 < A < 1$$

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

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

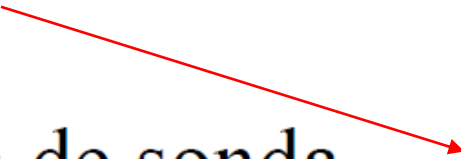
# Açık adresleme ile çarpışmaları çözmek

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).

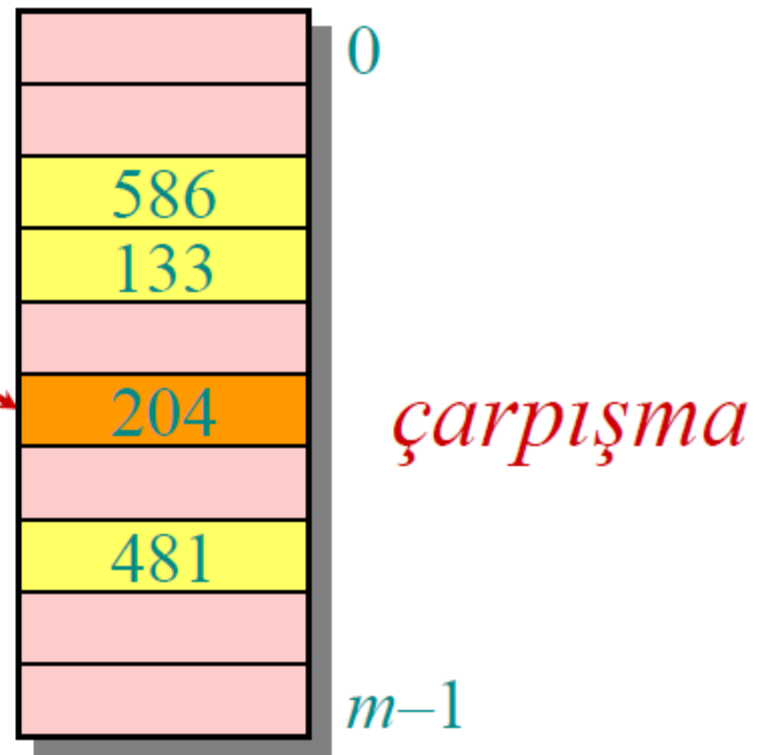


Başka bir ifadeyle  
doluyorsa tekrar  
kısımlama yapar  
boş bir yer arar.

# Açık adresleme ile çarpışmaları çözmek

Anahtarı  $k = 496$  araya yerleştirin:

0. Sonda  $h(496, 0)$

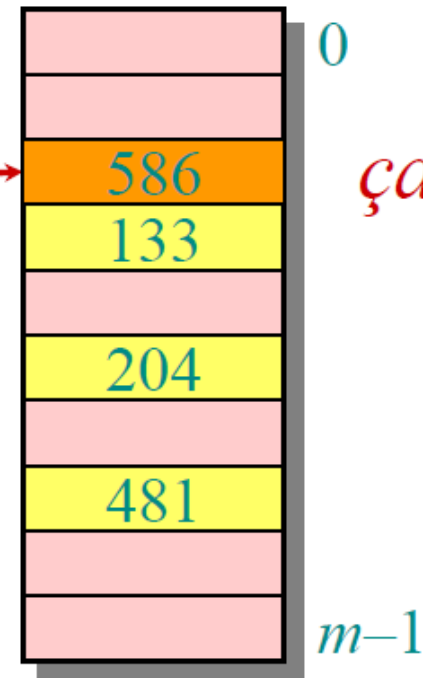


# Açık adresleme ile çarpışmaları çözmek

Anahtarı  $k = 496$  araya yerleştirin:

0. Sonda  $h(496, 0)$

1. Sonda  $h(496, 1)$



*çarpışma*

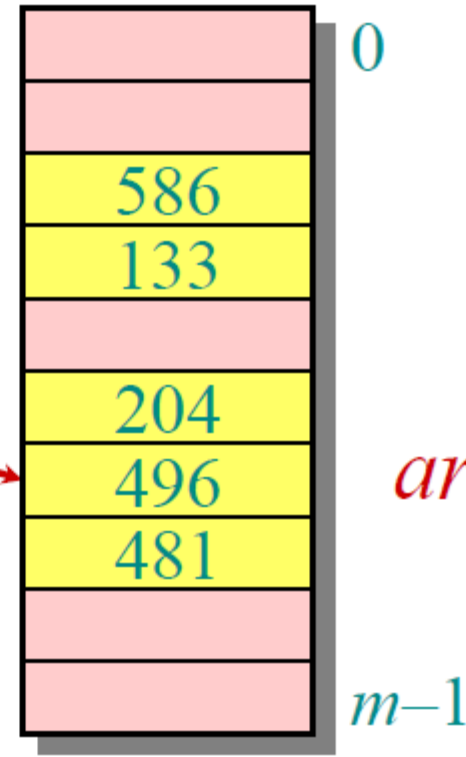
# Açık adresleme ile çarpışmaları çözmek

Anahtarı  $k = 496$  araya yerleştirin:

0. Sonda  $h(496, 0)$

1. Sonda  $h(496, 1)$

2. Sonda  $h(496, 2)$



# Açık adresleme ile çarpışmaları çözmek

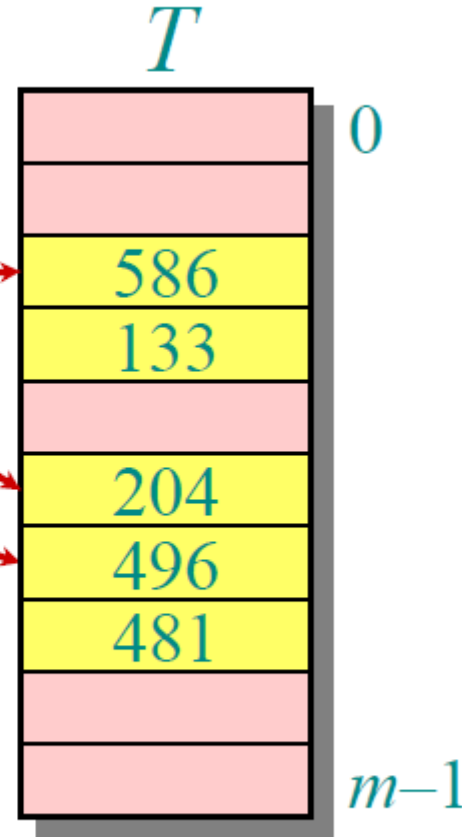
$k = 496$ : Anahtarı arama

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.



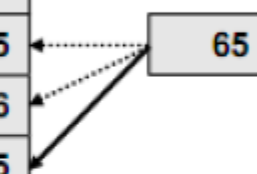
# Sonda stratejileri

- ◉ 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)
  - ◉ –  $h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod m$

# Çakışmanın giderilmesi Doğrusal Sondalama (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	-





# Çakışmanın giderilmesi Doğrusal Sondalama (Linear Probing)

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

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

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

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

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

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

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

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

---

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

## Doğrusal Sondalamanın 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.

## Çakışmanın giderilmesi İkinci Dereceden Sondalama (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:  $t = [h'(k)]$
- $h(k,i) = (t + c_1i + c_2i^2) \bmod m$

## Örnek - İkinci Dereceden Sondalama (Quadratic Probing)

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

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

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

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

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

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

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

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

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

## İkinci Dereceden Sondalama avantajları / dezavantajları

- Anahtar değerlerini linear probing metoduna göre daha düzgün dağıtır.
- Yeni eleman eklemeye tablo boyutuna dikkat edilmezse sonsuza kadar çalışma riski vardır.

# Çifte (Double) Kısımlama

$h_1(k)$  ve  $h_2(k)$ , gibi iki basit kırım fonksiyonu varsa, çifte kısımlama şu kırım 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.



# Çifte (Double) Kısımlama

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>

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

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

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

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

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

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

# Çifte Kısımlama Teoremin kanıtlanması

- 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...



# Çifte Kısımlama Teoremin kanıtlanması

Bu nedenle, beklenen sonda sayısı:

$$1 + \frac{n}{m} \left( 1 + \frac{n-1}{m-1} \left( 1 + \frac{n-2}{m-2} \left( \dots \left( 1 + \frac{1}{m-n+1} \right) \dots \right) \right) \right)$$

$$\leq 1 + \alpha (1 + \alpha (1 + \alpha (\dots (1 + \alpha) \dots)))$$

$$\leq 1 + \alpha + \alpha^2 + \alpha^3 + \dots$$

$$= \sum_{i=0}^{\infty} \alpha^i$$

$$= \frac{1}{1-\alpha} \cdot \square$$

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  $\alpha$  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.

# Kaynakça

Algoritmalar : Prof. Dr. Vasif NABİYEV, Seçkin Yayıncılık

Algoritmalara Giriş : Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, Palme YAYINCILIK

Algoritmalar : Robert Sedgewick , Kevin Wayne, Nobel Akademik Yayıncılık

M.Ali Akcayol, Gazi Üniversitesi, Algoritma Analizi Ders Notları

Doç. Dr. Erkan TANYILDIZI, Fırat Üniversitesi, Algoritma Analizi Ders Notları

<http://www.bilgisayarkavramlari.com>