4.3.3 Çırpı (Hashing)

→ bu bölümde O(1) zamanlı arayabileceğiz

```
>>> baskentler = {'Turkiye':'Ankara', 'Amerika':'Washington'}
>>> baskentler
{'Amerika': 'Washington', 'Turkiye': 'Ankara'}
>>> baskentler['Amerika']
'Washington'
>>> baskentler['Turkiye']
'Ankara'
>>> baskentler['Irak']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Irak'
```

→ kullanacağımız kavram: **çırpı** (hash)

Çırpı nedir?

→ todo

çırpı

→ elemanların kolleksiyondaki yerini yaklaşık olarak bilmeliyiz

→ her eleman olması gereken yerdeyse, **tek** karşılaştırma yeterli olur: O(1)

→ fakat tipik durum biraz daha farklıdır

çırpı tablosu

çırpı tablosu,

→ koleksiyondaki öğeleri

→ daha sonra kolayca erişebileceğimiz biçimde saklamak

→ saklandığı tablo

Slot,

→ tablonun her bir konumu.

→ 0 ile başlar ve

→ Slot0 biçimde isimlendirilir.

→ Başlangıçta hepsi boştur: None

çırpı tablosu

→ m = 11 iken

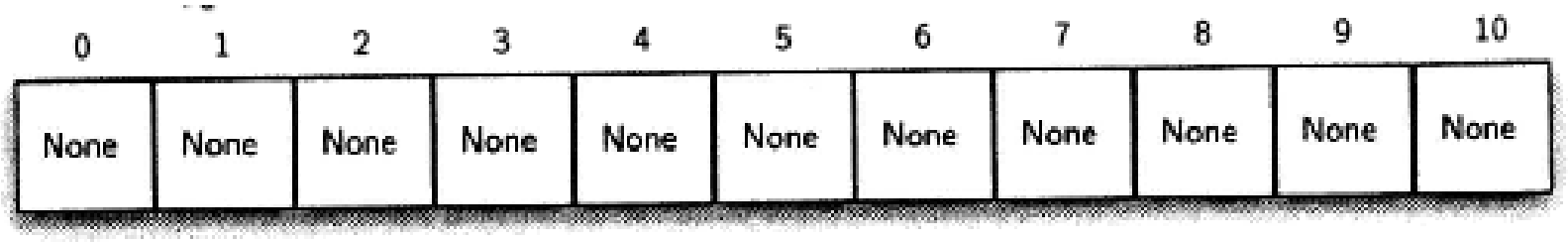

Figure 4.5: Hash Table with 11 Empty Slots

çırpı işlevi

çırpı işlevi: öğeleri slota haritalamada çırpı işlevi kullanılır.

→ koleksiyonumuz: [54, 26, 93, 17, 77, 31] olsun

→ koleksiyondaki herhangi bir elemanı al (ör. 54)

→ işlevden geçirerek slot numarasını elde et (ör. 4)

→ peki nasıl?

çırpı işlevi: modül kullanımı

- → basit olarak modül aritmetiği veya kalan hesabı kullanılabilir
- → tüm çırpı işlevlerinde modül alma (%) vardır
- → Çıktıyı 0...(m-1) arasına düşürmek
- → Ör. h(item) = item % m, burada m = 11

Item	Hash Value
54	10
26	4
93	5
17	6
77	0
31	9

Table 4.5: Simple Hash Function Using Remainders

çırpı tablosu

→ öğeler --> çırpı işlevi (h) --> slot numarası

→ tabloda slot numaralı yere öğeyi yerleştir

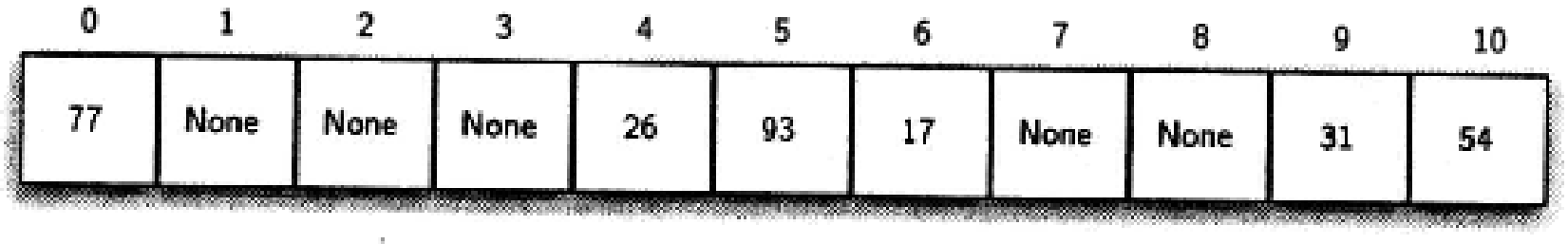

Figure 4.6: Hash Table with Six Items

yük etmeni

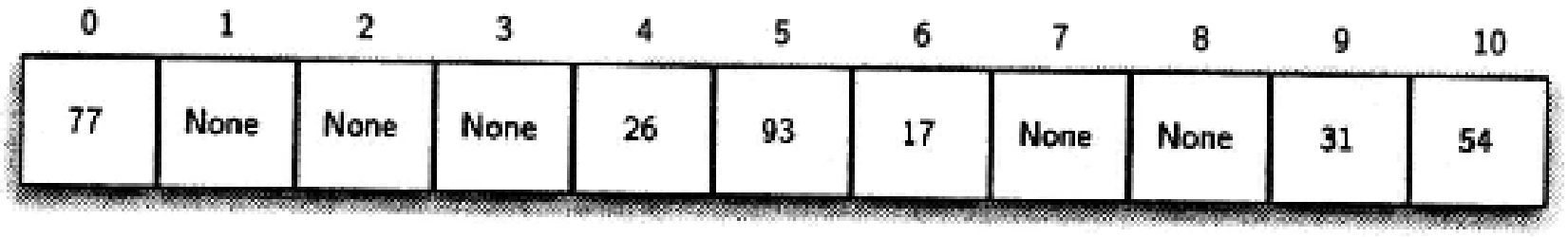

Figure 4.6: Hash Table with Six Items

→ tablonun sadece 6 / 11 elemanı dolu

→ doluluk miktarına / oranına yük etmeni denilir

→ lambda = (oge sayisi) / (tablo boyutu)

sabit zamanda erişim

çırpı yardımıyla arama sırasında

- → aranan elemanı çırpı işlevinden geçir
- → slot numarasını elde et
- → ilgili slota bak
- → O(1): sabit= aynı zamanda erişim

çakışma (collusion)

→ sabit zamanda erişim, her bir elemanın farklı slota yönlendirilmesini gerektirir (**unique**)

```
77 % 11 --> Slot0
44 % 11 --> Slot0
```

→ ne olacak?

→ bu duruma çakışma (collusion) diyoruz

4.3.3.1 Çırpı İşlevleri - SVT

mükemmel çırpı işlevi: çakışmanın oladığı durumdur.

- → yeni öğeler eklenmeyecekse sorun yok
- → keyfi elemanlarla çalışıyorsak mükemmel çırpı için sistematik bir yol yok!
- → bir yol, tablo boyutunu arttırmak, bu ise başka bir soruna neden olur
- → 9 haneli sosyal güvenlik numarası için 1 milyon slot???
- → 25 öğrenci için anormal boyutta bellek, pratik değil!

amaç

- → çakışmayı minimize edecek çırpı işlevini oluşturmak
- → hesabı kolay da olmalı
- → basit modül işleci yetmedi, iyileştirmeliyiz

folding yöntemi

→ telefon numarası:
436-555-4601

Yöntem1

1. ikili grupla:
   43-65-55-46-01
2. grupları topla:
   43+65+55+46+01
   $\Longrightarrow$ 210
3. modül al: 210 % 11 $\Longrightarrow$ 1(*slot*1)

Yöntem2

1. ikili grupla:
   43-65-55-46-01
2. sayıları ters çevir: 56
   $\Longrightarrow$ 65
3. grupları topla:
   43+56+55+64+01 $\Longrightarrow$ 219
4. modül al: 219 % 11 $\Longrightarrow$ 10(*slot*10)

mid-square yöntemi

1. karesini al: 44^2 $\Longrightarrow$ 1.936
2. ortadan iki hane seç: 93
3. modül al: 93 % 11 $\Longrightarrow$ 5(*slot*5)

özet tablo

Item	Remainder	Mid-Square
54	10	3
26	4	7
93	5	9
17	6	8
77	0	4
31	9	6

Table 4.6: Comparison of Remainder and Mid-Square Methods

karakter tabanlı çırpı işlevi

1. dizgi $\Longrightarrow$ *karakter* $\Longrightarrow$ *say*ß : cat $\Longrightarrow$ c $\Longrightarrow$ 99

```
>>> ord('c')
99
>>> ord('a')
97
>>> ord('t')
116
```

2. sayıları topla: 99 + 97 + 116 $\Longrightarrow$ 312

3. modül al: 312 % 11 $\Longrightarrow$ 4(*slot*4)

çizim

→ çizim

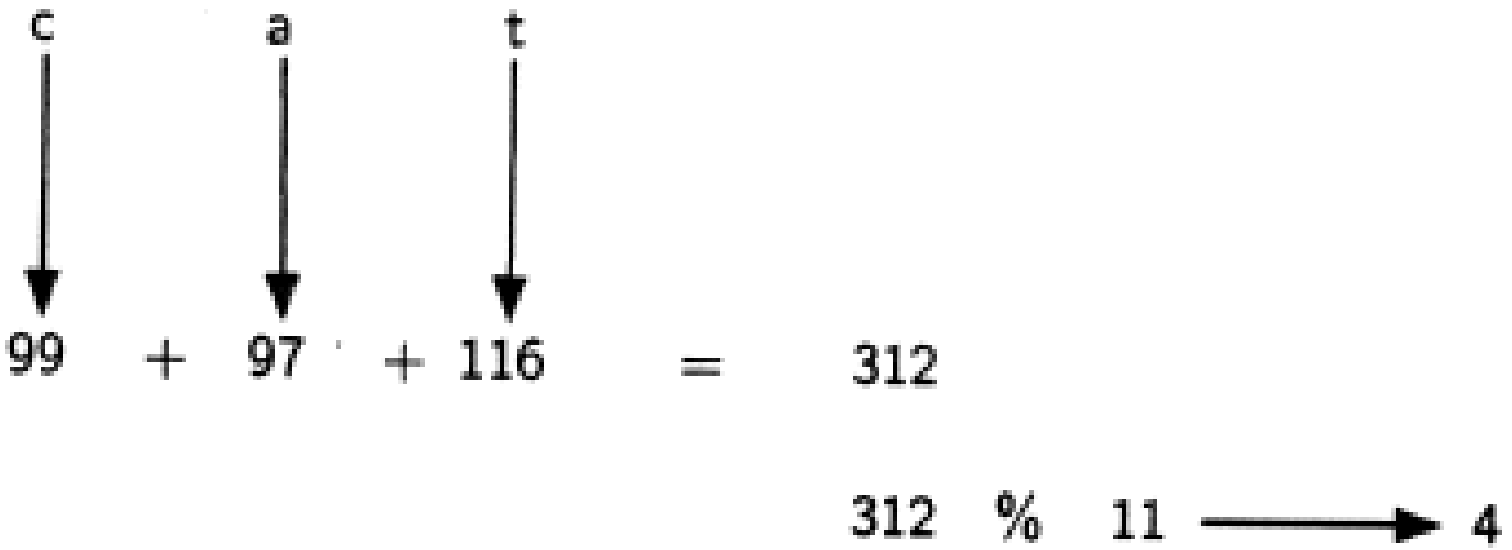

Figure 4.7: Hashing a String Using Ordinal Values

gerçekleme

→ gerçekleme

```
1       def hash(astring, tablesize):
2           sum = 0
3           for pos in range(len(astring)):
4               sum = sum + ord(astring[pos])
5
6           return sum%tablesize
7
8       def hash_weighted(astring, tablesize):
9           sum = 0
10          i = 1
11          for pos in range(len(astring)):
12              sum = sum + i * ord(astring[pos])
13              i = i + 1
14
15          return sum%tablesize
```

anagram durumu

- → cat , tac iken
- → hash değerleri?

anagram durumu

- → cat , tac iken
- → hash değerleri? aynı

anagram durumu

- → cat , tac iken
- → hash değerleri? aynı
- → bunu düzeltmek için ağırlıklandırmalı hashleme yapılabilir

ağırlıklandırılmış hashleme

→ ağırlıklandırılmış hashleme

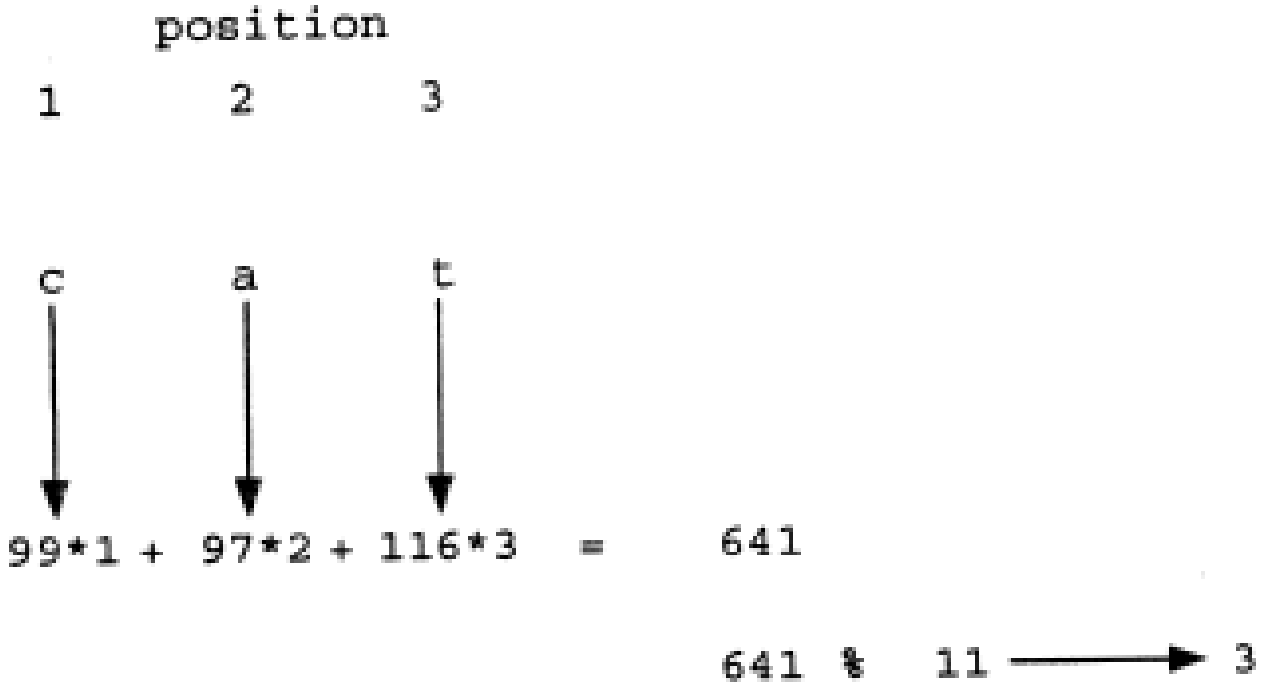

Figure 4.8: Hashing a String Using Ordinal Values with Weighting

mükemmel çırpı işlevini belirlemeye ne dersiniz?

→ bunlarla sınırlı kalmanız gerekmez

→ sınıf mevcudumuz = 50 kişi

→ girdi: ad soyad (turkce karakter kullanmayalım, şimdilik)

→ çıktı: 1-50 arası slot numarası

başarım ölçütü

→ başarım ölçütü:

1. 100 * (hash işlevinizin üretebildiği farklı slot sayısı) / 50
2. hesap süresi

```
start = time.clock()

your_hash(str, 50)

end = time.clock()

gecen_sure = end - start
```