4.3.3.3 HashTable sınıfı

→ çırpı tablosu sözlük gibi davranır

```
>>> baskentler = {'Turkiye':'Ankara', 'Amerika':'Washington'}
>>> baskentler
{'Amerika': 'Washington', 'Turkiye': 'Ankara'}
>>> baskentler['Amerika']
'Washington'
>>> baskentler['Turkiye']
'Ankara'
```

→ sözlük, anahtar:değer çiftleri

HashTable sınıfı

- her bir slot, bir elemana (veya anahtar) ve
- ona karşılık gelen veriye (veya değere) sahip olacak
- çakışmalar, saklama yöntemince idare edilir
- burada kullanılan yöntem aramada da tekrarlanmalıdır

HashTable SVT

HashTable(size):

→ yeni çırpı tablosu üret.

→ size boyutlu.

→ indisler/slotlar: 0 - (size-1) ile isimlensin

devam

store(item, data):

→ item: anahtar, data: değer

→ anahtarla çırpımın sonucundaki slota değeri (ve anahtarı) koy

search(item):

→ anahtar ile ilişkili veriyi (varsa) döndürür,

→ yoksa None döndürür

HashTable

→ gerçekleme

```
1       class HashTable:
2           def __init__(self,size):
3               self.slots = [None] * size
4               self.data = [None] * size
```

→ paralel iki liste: slots, data

store

→ gerçekleme

```
1         def store(self,item,data):
2           hashvalue = self.hashfunction(item,len(self.slots))
3
4           if self.slots[hashvalue] == None:
5             self.slots[hashvalue] = item
6             self.data[hashvalue] = data
7           else:
8             nextslot = self.rehash(hashvalue,len(self.slots))
9             while self.slots[nextslot] != None:
10              nextslot = self.rehash(nextslot,len(self.slots))
11
12            self.slots[nextslot]=item
13            self.data[nextslot]=data
14
15        def hashfunction(self,item,size):
16                return item%size
17
18        def rehash(self,oldhash,size):
19            return (oldhash+1)%size
```

açıklama

- → çırpma basit modül
- → yeniden çırpmada skip=1
- → s2: çırp
- → s4–s6: slot boşsa koy
- → s7–s13: boş slota rastlayıncaya kadar yeniden çırp, bulduğunda koy

search

→ gerçekleme

```
1           def search(self,item):
2             startslot = self.hashfunction(item,len(self.slots))
3
4             data = None
5             stop = False
6             found = False
7             position = startslot
8             while self.slots[position] != None and  \
9                                  not found and not stop:
10               if self.slots[position] == item:
11                 found = True
12                 data = self.data[position]
13               else:
14                 position=self.rehash(position,len(self.slots))
15                 if position == startslot:
16                     stop = True
17            return data
18
19          def __getitem__(self,item):
20              return self.search(item)
21
22          def __setitem__(self,item,data):
23              self.store(item,data)
```

açıklama

- → s2: çırp
- → s10–s12: slotta varsa döndür
- → s13–s16: slotta yoksa, yeniden çırparak aramaya devam et
- → s16: yeniden çırpmalarda başladığın noktaya döndüysen bulunamadı demektir

demo

→ demo

```
1  from listing_4_14_16 import *
2  H = HashTable(11)
3  H[54] = "cat",  H[26] = "dog"
4  H[93] = "lion", H[17] = "tiger"
5  H[77] = "bird", H[31] = "cow"
6  H[44] = "goat", H[55] = "pig"
7  H[20] = "chicken"
8  print H.slots
9  # [77, 44, 55, 20, 26, 93, 17, None, None, 31, 54]
10 print H.data
11 # ['bird', 'goat', 'pig', 'chicken', 'dog', 'lion', 'tiger', None, None, 'cow
12 print H[20] # 'chicken'
13 print H[17] # 'tiger'
14 print H[1]  # None
```