VERİ YAPILARI

HASH TABLOLARI

DR. GÜNAY TEMÜR

İÇERİK

Hash Tabloları

Hash Fonksiyonu

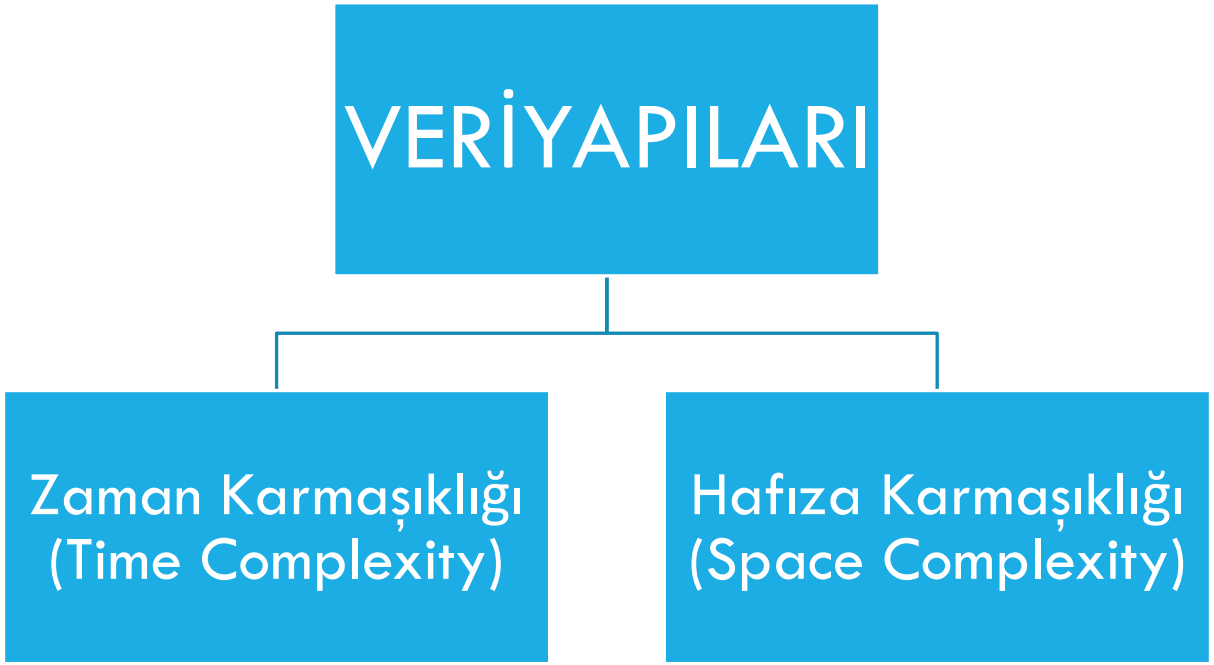
Çakışma (Collision) ve Çözümler

- Ayrık Zincirleme (Separate Chaning)
- Açık Adresleme (Open Adressing)
  - Doğrusal Ölçüm (Linear Probing)
  - Karesel Ölçü (Quadratic Probing)

VERİ YAPILARI

Bilgi teknolojileri dünyasında bir problemin çözümü için;

Veri Yapısı modellerinin kullanılmasında **2 temel amaç** vardır.



Ya en hızlı algoritmayı yazacağız;

Yada minimum hafıza kullanacağız;

VERİ YAPILARI

VERİ YAPILARI VE VERİ YAPILARI MODELLERİ

Minimum Hafıza kullanımı veya **minimum Zamanda**

- Ekleme (Yeni Kayıt)
- Sıralama
- Arama (Üzerinde İşlem)
- Silme

problemlerini çözmeyi hedeflerler.

PROBLEM TANIMI

Günümüz teknolojisini ele aldığımızda, elde edilen verilerin mutlaka bir şekilde **depolanması** gerekmektedir.

Örnek: Öğrenciler mutlaka bir veri tabanına kaydedilmeli.

Öğrenci No
Adı
Soyadı
Doğrum Tarihi
Memleketi
Bölümü

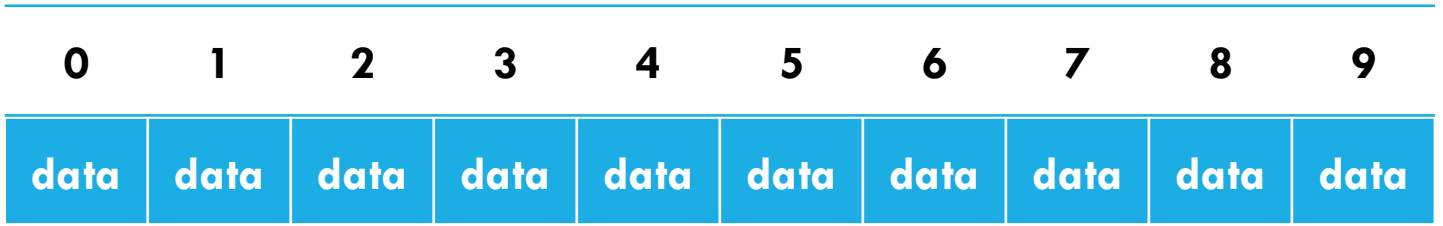
EKLEME PROBLEMİ

PROBLEM TANIMI

İki tip hafıza erişim stratejisi mevcuttur:

1. Sıralı erişim

2. Direk erişim

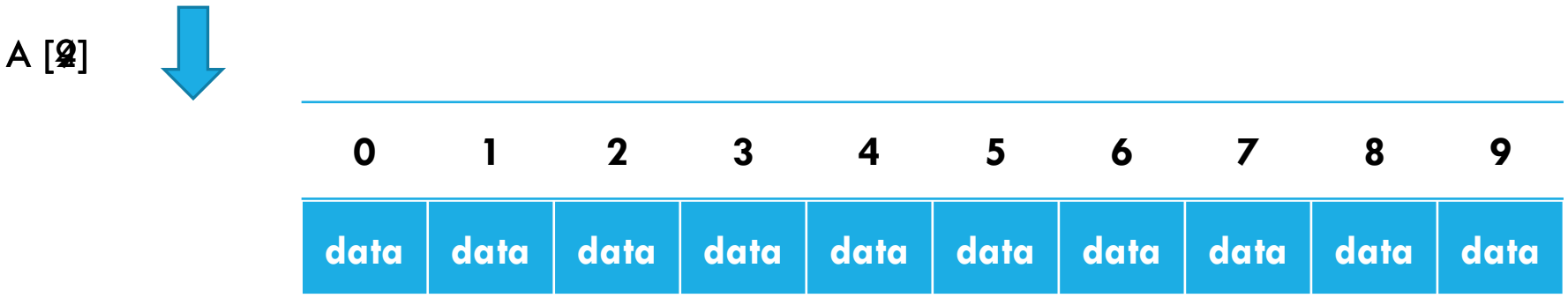


Sıralı erişimde

- temel işlem, **baştan sona** **tüm anahtarları karşılaştırmaktır**.
- Bir anahtarın tablo içerisinde bulunduğu pozisyona ulaşıncaya kadar arama işlemine **devam edilir**.

Direk erişimde

temel işlem, verilen **anahtar indis numarasında** istenilen datanın olup olmadığını **karşılaştırmaktır**.



Direk erişim; **zamanı en verimli** olan hafızaya **erişim stratejisidir**.

PROBLEM TANIMI

Tüm verilerimizi biz diziye kaydedelim.

Ekleme işlemi için benzersiz bir **ID** ye ihtiyacımız var.

TC kimlik No: 111 111 11 111

Öğrenci No: 171001230

Aday No:

Sicil No:

gibi

11 Haneli TC kimlik No düşünüldüğünde

- 80 000 000 insanı kaydetmek için;
- 99 999 999 999 indisli biz dizi oluşturabiliriz.

9 Haneli Öğrenci No düşünüldüğünde

- 50 000 öğrenci için
- 999 999 999 indisli bir dizi oluşturabiliriz.

PROBLEM TANIMI

171001230	171001231	.....	181001232	.....	211001533
Öğrenci No	Öğrenci No	Öğrenci No	Öğrenci No	Öğrenci No	Öğrenci No
Adı	Adı	Adı	Adı	Adı	Adı
Soyadı	Soyadı	Soyadı	Soyadı	Soyadı	Soyadı
Doğrum Tarihi	Doğrum Tarihi	Doğrum Tarihi	Doğrum Tarihi	Doğrum Tarihi	Doğrum Tarihi
Memleketi	Memleketi	Memleketi	Memleketi	Memleketi	Memleketi
Bölümü	Bölümü	Bölümü	Bölümü	Bölümü	Bölümü

İdeal Çözüm: HASH TABLOLARI

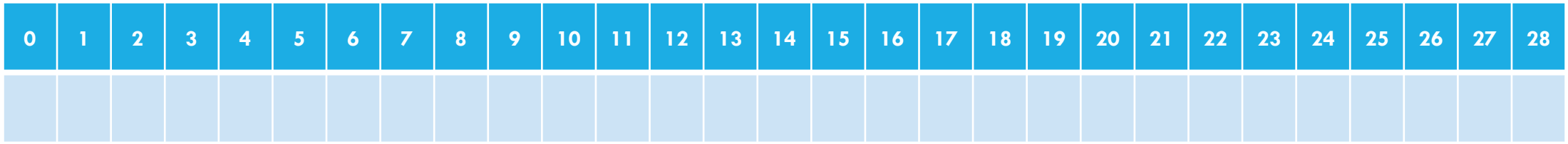
Bu tablolar (diziler), bir koleksiyondaki bir öğeyi verimli bir şekilde saklama veya arama sorununu çözmek için tasarlanmıştır.

Hash tablo veri yapısı modeli ile veri arama, ekleme ve silme işlemleri **ortalama olarak sabit zamanda** (O(1)), verimli bir biçimde gerçekleştirilir.

Programlanmaları da diziler kadar kolaydır.

HASH TABLOLARI (devam…)

İdeal bir Hash Tablosu veri yapısı modeli, içerisinde elemanlar barındırabilecek **sabit bir diziden oluşur**. Dizi boyutu yerleştirilecek eleman sayısından biraz fazla veya maksimum 2 katı olabilir.



- Dizinin boyutu ***TabloBoyutu*** olup,
- Diziye eklenecek elemanlar 0'dan ***(TabloBoyutu-1)***'e kadar olan indekslerde saklanırlar.

HASH TABLOLARI (devam…)

Dizi içerisindeki elemanlar, indis hesaplamasında kullanılacak **anahtar (key)** isimli özel bir üye bulundurmak **zorundadırlar**.

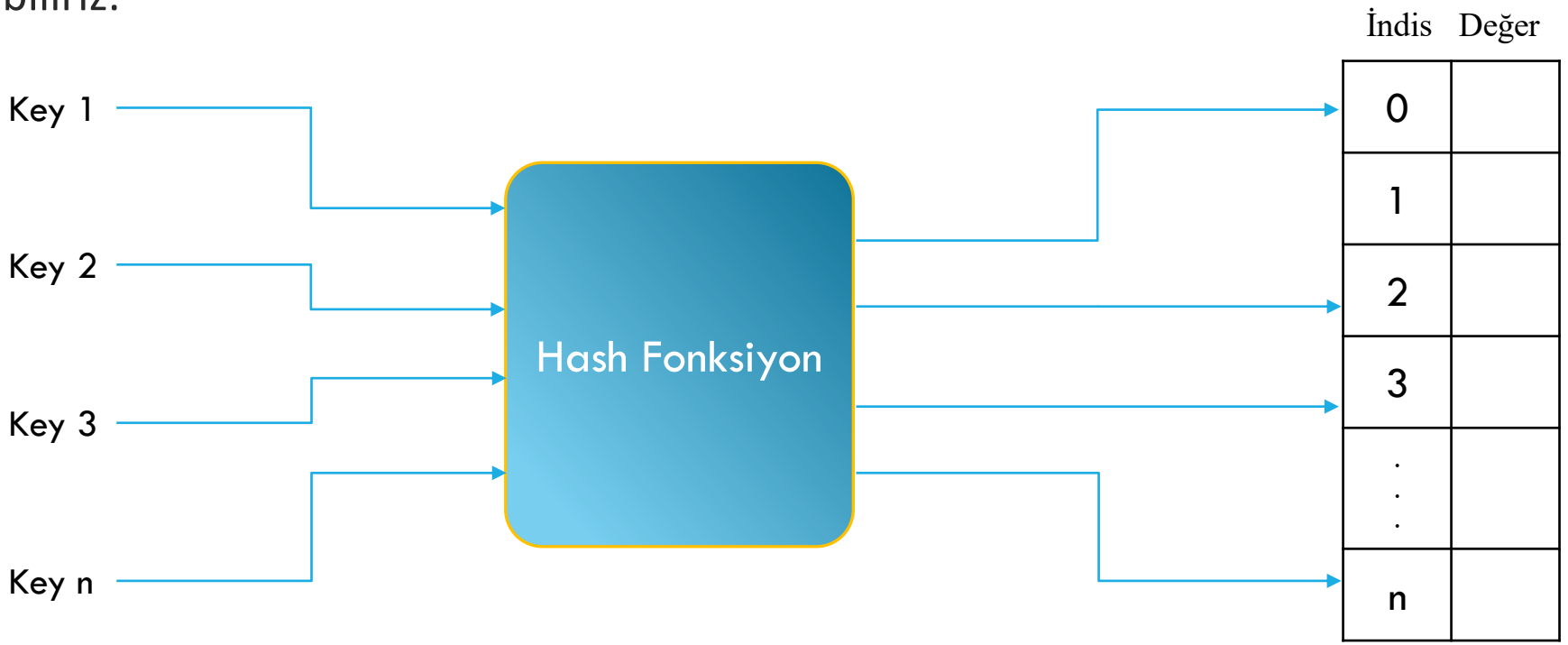
- Anahtar; integer veya **string** bir değer olabilir.
- Temel olarak hash tablosunun 2 ana bileşeni vardır: Hash fonksiyonu ve dizi.

HASH TABLOLARI (devam…)

Anahtardan→Indekse dönüştürme işlemine **Hashing**, bu işi yapan fonksiyona ***Hash fonksiyonu*** denir.

HASH TABLO

Şekilde görüldüğü gibi, bir KEY'i hash fonksiyonumuza iletebilir, hash kodunu alabilir, bu kod ile kayıtları ekleme veya sorgulama işlemi gerçekleştirebiliriz.



HASH TABLO

- Hash' in nasıl çalıştığını anlamak için bir örnek görelim.
- Dizi anahtarlarının bir listesini dizi değerlerine eşlemek istediğimizi varsayalım. Diyelim ki Tablo 1'deki verileri Hash tablosunda depolamak istiyoruz.

Tablo 1

KEY	VALUE
1111	Cem
2222	Ada
6666	Sibel
4444	İpek

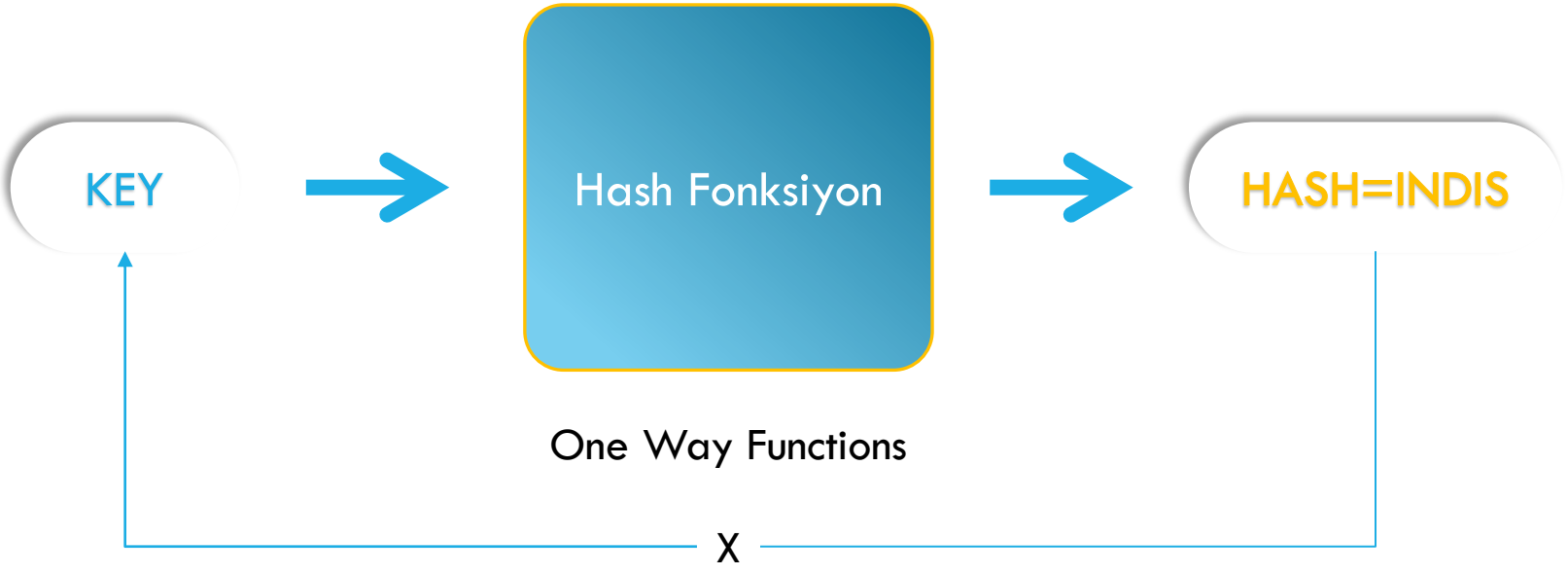
HASH FONKSİYONLARI

- Anahtarın **dizideki pozisyonunu** yani **indisini** belirlemek için kullanılır.
- Herhangi bir anahtara ait şifrelenmiş veya kısaltılmış değer üretmek için kullanılan bir **işlev** veya **algoritmadır**.
- Bir hash fonksiyonunun sonucu, ***hash değeri*** veya basitçe ***hash*** olarak adlandırılır.

HASH FONKSİYONLARI

HASH fonksiyonlarının en iyi özellikleri **Tek yönlü** algoritmalar olmalarıdır.

Yani: Üretilen HASH (INDIS) değerinden geri dönüşüm **mümkün değildir**.



***Hash fonksiyonları verimli bir şekilde **hesaplanabilir** olmalıdır. Gerçek uygulamalarda, algoritmamızın bir hash fonksiyonundan hash değerini hesaplaması uzun zaman alıyorsa, o zaman hashing amacını kaybederiz.

HASH FONKSİYONLARININ TÜRLERİ

(Index Mapping Method) Eş İndis yöntemi

(Mid square method) Karesel Orta Değer yöntemi

(Digit folding method) Basamak Değer yöntemi

(Multiplication method) Çarpım yöntemi

(Division method) Mod Alma yöntemi

Index Mapping

InterviewBit

Hash Function:
Division Method

InterviewBit

Table_Length= 6
Values to be hashed = 6, 9, 19, 22, 29

hash(Key) = key % Table_Length

_> hash(6) = 6 % 6 = 0 => Place key **6** in **0th** position
_> hash(9) = 9 % 6 = 3 => Place key **9** in **3rd** position
_> hash(18) = 19 % 6 = 1 => Place key **19** in **1st** position
_> hash(21) = 22 % 6 = 2 => Place key **22** in **2nd** position
_> hash(29) = 29 % 6 = 5 => Place key **29** in **5th** position

Hash Function:
Mid Square Method

InterviewBit

Key = 3101
Table_Size = 2000

hash(Key) = middle numbers from key * key value

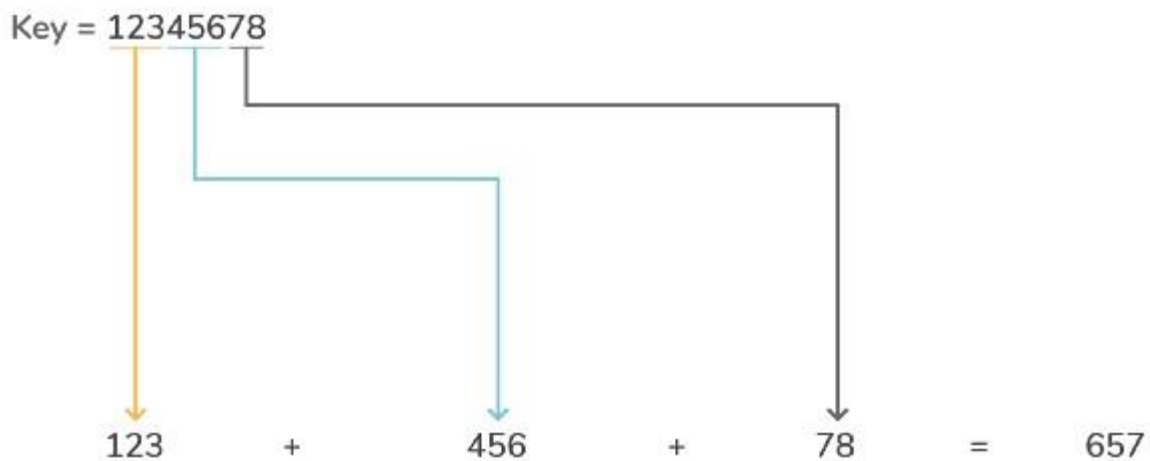
=> hash(3101):
3101 * 3101 = 9 6 1 6 2 0 1

Middle Number = 162

Place record of key 3101 at 162nd position in hash table

Hash Function:
Digit Folding Method

InterviewBit



Place record of key **12345678** *at* **657th** *position in the hash table*

Hash Function:
Using Multiplication Method

InterviewBit

Key = 50
Assume c = 0.81, where 0 < c < 1
Assume Table_Size = 1000

hash(Key) = floor(Table_Size * fractional(k * c))

hash(50) = floor(1000 * fractional(50 * 0.81))

50 * 0.81 = 40.5 => Fractional Part = x - floor(x)
= 40.5 - floor(40.5)
= 40.5 - 40
= 0.5

hash(50) = floor(1000 * 0.5) = floor(500) = 500

Place record of key 50 at 500th position in hash table

MOD ALMA YÖNTEMİ

- Dizinin **eleman sayısı N** olsun
- Fonksiyon **f(x)**, *x anahtarını* **0 ve N–1** arasındaki bir indise dönüştürür
- Örneğin, **N=15 ise**, anahtar 0 ve MAX_INT arasında olup, **Hash fonksiyonu** aşağıdaki gibi olabilir:

$$f(x) = x \% 15$$

MOD ALMA YÖNTEMİ

f(x) = x % 15 **ise f(x) ?**

if x =	25	129	35	2501	47	36
f(x) =	10	9	5	11	2	6

Anahtarların diziye **yerleşimi** aşağıdaki gibidir:

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

- Silme, ekleme ve arama **işlem karmaşıklığı O(1),** ancak **bir problem var???**

MOD ALMA YÖNTEMİ

Soru: Eğer x = 65 anahtarını eklemek istersek ne olur?

x = 65

f(x) = 5

Anahtarların diziye **yerleşimi**:

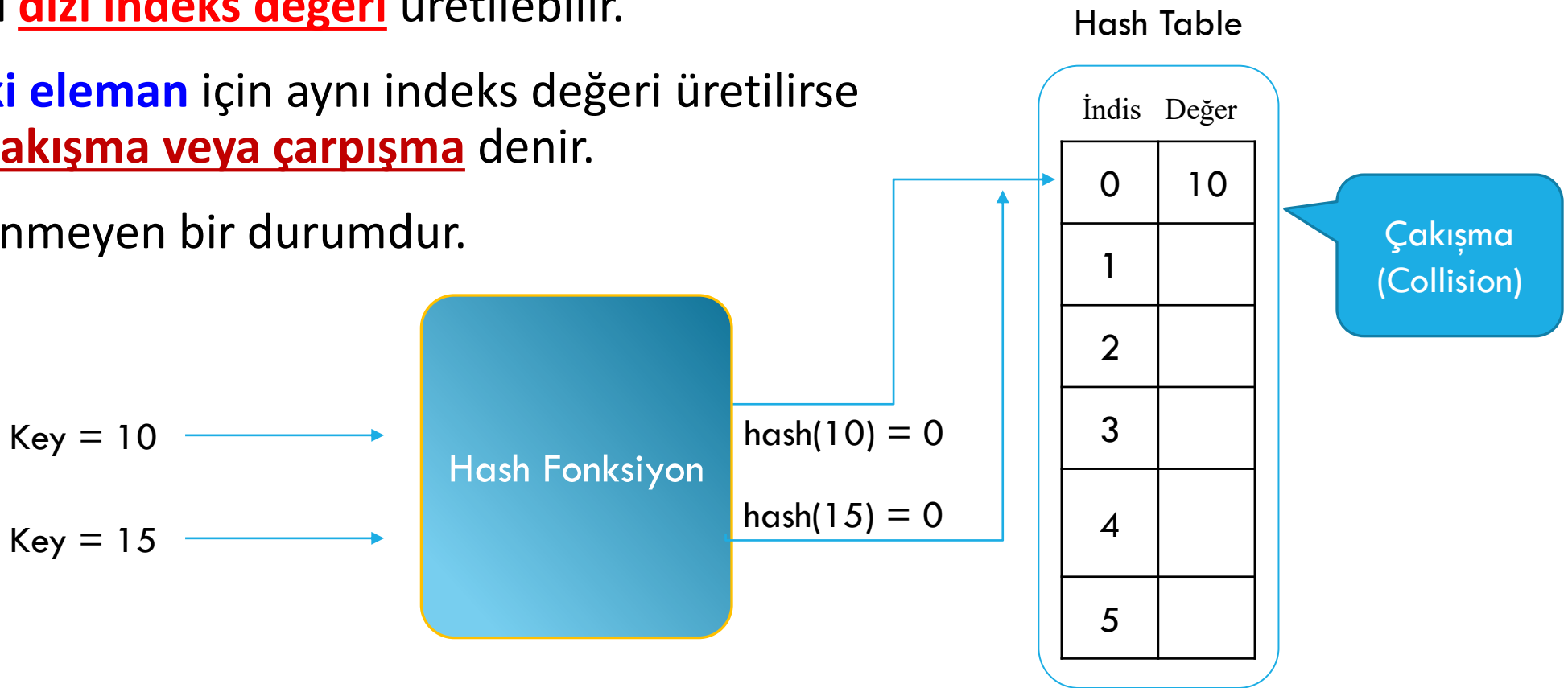
```
0   1   2   3   4   5   6   7   8   9  10   11  12  13  14
_   _  47   _   _  35  36   _   _ 129  25 2501   _   _   _
                       65(?)
```

- 35 ve 65 anahtarları için f(x) fonksiyonundan **aynı indis değeri** döndü.

  Bu duruma; ***Çakışma** (**collision**) denir.*

Çakışma (Collision)

- Hashing işlemi sonucunda farklı anahtarlara sahip iki eleman, **aynı dizi indeks değeri** üretilebilir.
- Eğer **en az iki eleman** için aynı indeks değeri üretilirse bu duruma **çakışma veya çarpışma** denir.
- Çakışma istenmeyen bir durumdur.



Çakışma (Collision)

- **Çakışmayı çözmek** için **iki yöntem** vardır:
    1. **Ayrık zincirleme (Separate chaning)**
    2. **Açık adresleme (Open adressing)**
        I. **Doğrusal ölçüm (Linear probing)**
        II. **Karesel ölçüm (Quadratic probing)**

Ayrık Zincirleme Çözümü

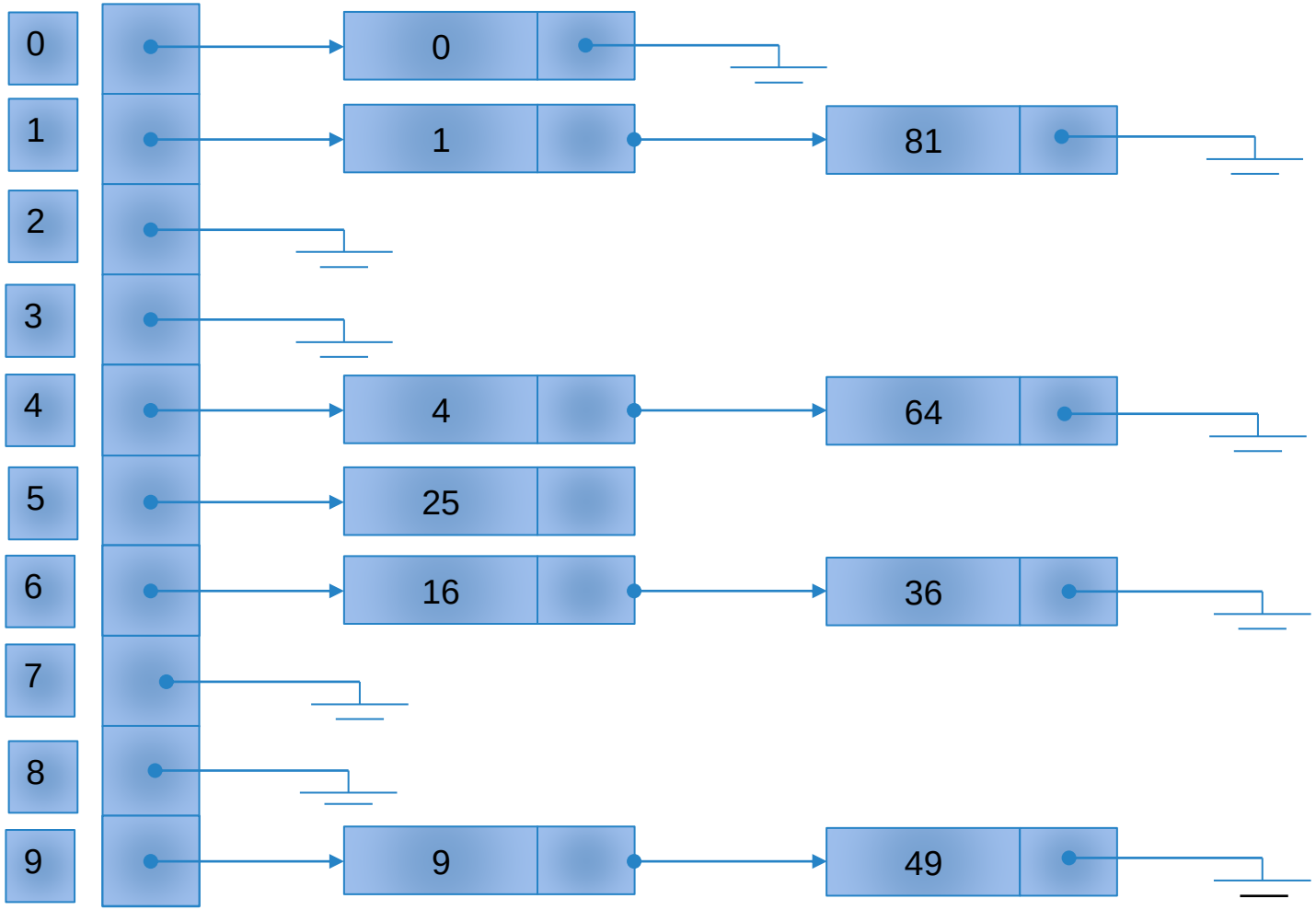
- **Aynı indis pozisyonuna** gelen kayıtlar **bağlı listelerle** gösterilir.
- **Çakışma meydana gelirse** ikinci eleman bir bağlı liste ile **birinci elemana bağlanır**.
- Bağlı listeler tek veya çift yönlü olabilir.

Ayrık Zincirleme Çözümü (devam...)

Anahtarlar:

0, 1, 4, 9, 16, 25, 36, 49, 64, 81

hash(key) = key % 10



Ayrık Zincirleme Çözümü (devam…)

- **Avantajları**
  - Basit ***çakışma çözümü*** (bağlı liste üzerinde arama)
  - Hash tablosunun maksimum eleman sayısından daha fazla eleman eklenebilir.
- **Dezavantajları**
  - Tablonun bazı kısımları hiç kullanılmamaktadır.
  - Bağlı listeler uzadıkça arama ve silme işlemleri için gereken zaman uzamaktadır.
  - Dizi haricinde ekstra veri yapısı olan bağlı liste kullanılır.

Açık Adresleme Çözümü

- Açık adresleme çözümünde tüm elemanlar **aynı hash tablosunda (dizide)** saklanırlar.
- Çakışma meydana geldiğinde **alternatif boş indisler** denenir.
  - Denenecek indisler $h_0(x)$, $h_1(x)$, $h_2(x)$, …
- **Genel mantık** aşağıdaki gibidir:
  - $h_i(x) = (hash(x) + f(i))$ mod *TabloBoyutu*, with $f(0) = 0$.
  - f fonksiyonu bir **çakışma (Collision) çözüm stratejisidir**.

Açık Adresleme Çözümü (devam…)

1. Doğrusal Ölçüm (Linear Probing)
2. Karesel Ölçüm (Quadratic Probing)

Doğrusal Ölçüm (Linear Probing)

- Çakışma meydana geldiğinde, **doğrusal arama mantığıyla**, **uygun boş yerler** **sırayla aranırlar**.
- f doğrusal bir fonksiyon olup, ***f(i) = i***
- Sırayla deneme işlemi gerçekleştirilir.
- Hash tablosunun **sonuna gelindiyse**, başa dönülür.
  - Döngüsel Kuyruk mantığında veya döngüsel dizi

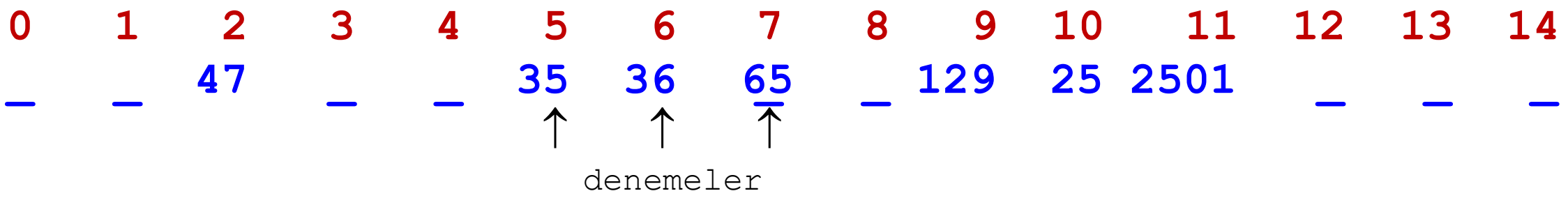
Doğrusal Ölçüm (Linear Probing) (devam…)

- *TabloBoyutu = 15.*
- **65 elemanını** eklemek istiyoruz, ancak **5 numaralı indiste** zaten 35 elemanı var.
- ***Çakışma*** gerçekleşti, ne yapacağız?

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

Doğrusal Ölçüm (Linear Probing) (devam…)

- **Çözüm:** 65 Ekle



- Toplam **3 deneme** yaptık ve uygun yeri bulduk.

Soru: 29'u nereye ekleyeceğiz?

Doğrusal Ölçüm (Linear Probing) (devam…)

- **Çözüm:** 29 Ekle

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
_	_	47	_	_	35	36	65	_	129	25	2501	_	_	29

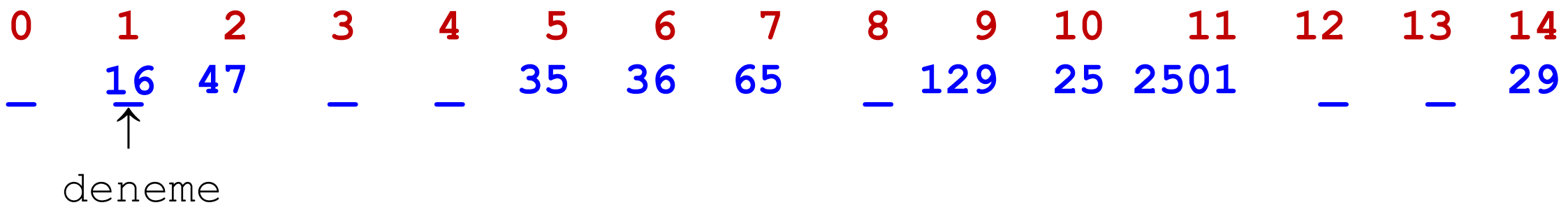
↑
deneme

- Toplam **1 deneme** yaptık ve uygun yeri bulduk.

Soru: 16'yı nereye ekleyeceğiz?

Doğrusal Ölçüm (Linear Probing) (devam…)

- **Çözüm:** 16 Ekle

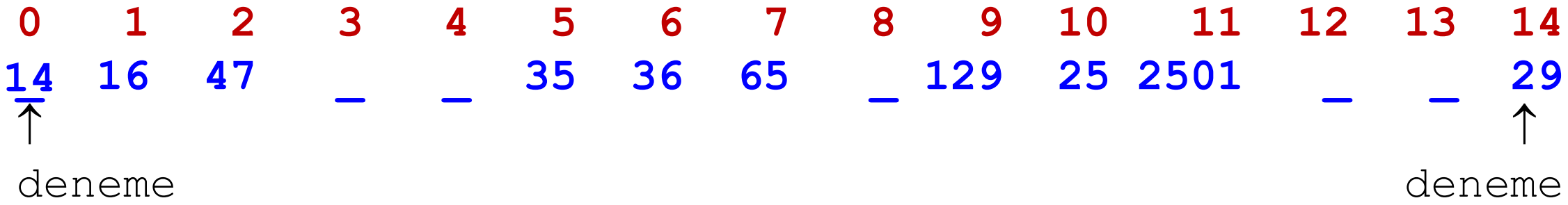


- Toplam **1 deneme** yaptık ve uygun yeri bulduk.

Soru: 14'ü nereye ekleyeceğiz?

Doğrusal Ölçüm (Linear Probing) (devam…)

- **Çözüm:** 14 Ekle

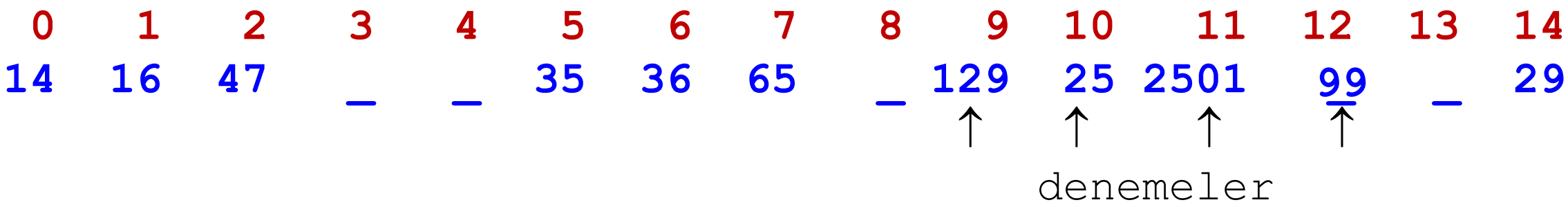


- Toplam **2 deneme** yaptık ve uygun yeri bulduk.

Soru: 99'u nereye ekleyeceğiz?

Doğrusal Ölçüm (Linear Probing) (devam…)

- **Çözüm:** 99 Ekle



- Toplam **4 deneme** yaptık ve uygun yeri bulduk.

Doğrusal Ölçüm (Linear Probing) (devam…)

Arama İşlemi

- Ekleme işlemindeki benzer algoritma mantığı ile arama işlemi gerçekleştirilir.
  1. Aranacak anahtar, hash fonksiyonuna geçirilir ve hash **tablo indisi bulunur**.
  2. İndis değerinde **eleman olup olmadığı** kontrol edilir. Eleman **yoksa**, bulma işlemi **başarısız** demektir. İndiste eleman varsa, **elemanın anahtarı ile aranan anahtar karşılaştırılır**.
     i. Eğer anahtarlar eşitse, çıkılır.
     ii. Anahtarlar eşit değilse, **bir sonraki indise geçilir**.
     iii. 2 numaralı adım baştan işletilir.

Örnek Soru

Doğrusal Ölçüm (Linear Probing) (devam…)

Silme İşlemi

- H(x) = x % 10
- Ekle 47, 57, 68, 18, 67
- Bul 68
- Bul 10
- Sil 47
- Bul 57

PROBLEM

47'yi silersek ne olur?

57'yi bulamayız. **Nasıl silmeliyiz?**

0	18
1	67
2	
3	
4	
5	
6	
7	~~47~~
8	57
9	68

ÇÖZÜM: Modu alınamayacak (Key olamayacak!) bir değer bulun.

Doğrusal Ölçüm (Linear Probing) (devam…)

Silme İşlemi Çözüm

- Tembel Silme (Lazy Deletion)
- Her dizi hücresinin **3 durumu** tutulur:
  - ***Aktif (Dolu)***
  - ***Boş (Empty) (Prog. Diline göre null veya 0)***
  - ***Silindi (Deleted)***
- Arama ve silme işlemlerinde
  - Sadece (**durum = Boş**) ise işlemi sonlandır. Silindi durumunda **sonraki elemanla devam et**.

0	**18**
1	**67**
2	
3	
4	
5	
6	
7	**-1**
8	**57**
9	**68**

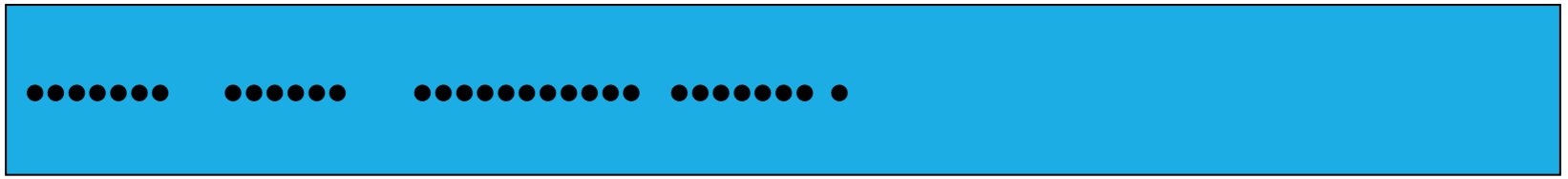
Doğrusal Ölçüm (Linear Probing) (devam…)

Kümelenme (Cluster) Problemi

Hash tablosu yeterince büyük olduğunda mutlaka boş bir hücre bulunabilir.

Kayıtların yığın şeklinde toplanmasına yani kümelenmeye neden olabilir.

Arama işlemi genelde küme içerisinde gerçekleşir ve kümeye eklenir.



Karesel Ölçüm (Quadratic Probing)

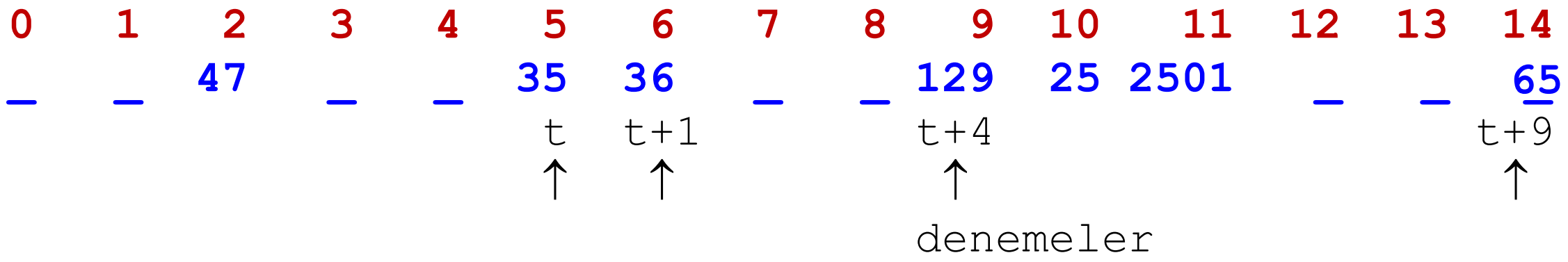
Kümeleme problemini önlemek için geliştirilmiştir.

Genel mantık aşağıdaki gibidir:

- $h_i(x) = (hash(x) + f(i))$ mod *TabloBoyutu*, with $f(0) = 0$.
- *f* fonksiyonu is the **çakışma (Collision) çözüm stratejisidir**.
- f karesel bir fonksiyon olup, $f(i) = i^2$

Karesel Ölçüm (Quadratic Probing) (devam…)

- **Çözüm:** 65 Ekle

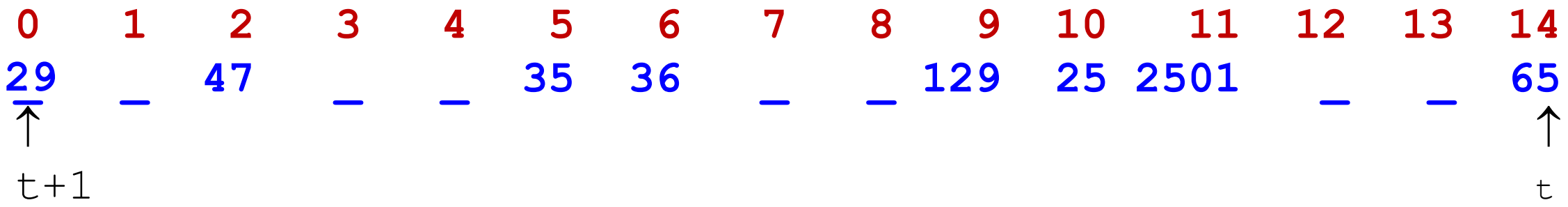


- Toplam **4 deneme** yaptık ve uygun yeri bulduk.

Soru: 29'u nereye ekleyeceğiz?

Karesel Ölçüm (Quadratic Probing) (devam…)

- **Çözüm:** 29 Ekle



- Toplam **2 deneme** yaptık ve uygun yeri bulduk.

Özet

- **Ekle**
  - Hücre: Boş veya Silindi — H indise Ekle VE *hücre = Aktif*
  - Hücre: Aktif — H = (H + 1) mod N
- **Bul / Ara**
  - Hücre: Boş — BULUNAMADI
  - Hücre: Silindi — H = (H + 1) mod N
  - Hücre: Aktif — if key == key in hucre -> BULDU; else H = (H + 1) mod N
- **Sil**
  - Hücre: Aktif; key != key — H = (H + 1) mod N
  - Hücre: Aktif; key == key — SİL; *hücre = Silindi*
  - Hücre: Silindi — H = (H + 1) mod N
  - Hücre: Boş — BULUNAMADI

SORU VE CEVAPLAR?

YARARLANILAN KAYNAKLAR

- **Web**
  - https://crackfaang.medium.com/intro-to-hashing-in-c-c0882b0c53b2
- **Ders Kitabı:**
  - Data Structures through JAVA, V.V.Muniswamy
- **Yardımcı Okumalar:**
  - Data Structures and Algorithms in Java, Narashima Karumanchi
  - Data Structures, Algorithms and Applications in Java, Sartaj Sahni
  - Algorithms, Robert Sedgewick
  - Yrd. Doç. Dr. Deniz KILINÇ, Celal Bayar Üniversitesi-Sunum üzerinde ekleme ve değişiklikler yapılmıştır.