Veri Yapıları

Öğr.Gör.Dr.Günay TEMÜR
Düzce Üniversitesi

Hash Tabloları ve Fonksiyonları

- Giriş
- Hash Tabloları
- Hash Fonksiyonu
- Çakışma (Collision)
- Ayrık Zincirleme Çözümü
- Linear Probing Çözümü
- Quadratic Probing Çözümü

Giriş

- Bilgi getiriminde **iki tip** bilgi arama ve erişim stratejisi mevcuttur:
  1. Sıralı erişim
  2. Direk erişim (**Nasıl yaparız?**)

Nasıl Yaparız?

- Sıralı erişimde, temel işlem, **baştan sona** **tüm anahtarları karşılaştırmaktır**.
- Bir anahtarın tablo içerisinde bulunduğu pozisyona ulaşıncaya kadar arama işlemine **devam edilir**.
- **Hash fonksiyonuyla** aranan **anahtar elemana doğrudan** erişilebilmektedir.
- Hash fonksiyonu, **bir anahtar bilgisinin** **tabloda bulunduğu indeksi** hesaplamaktadır.

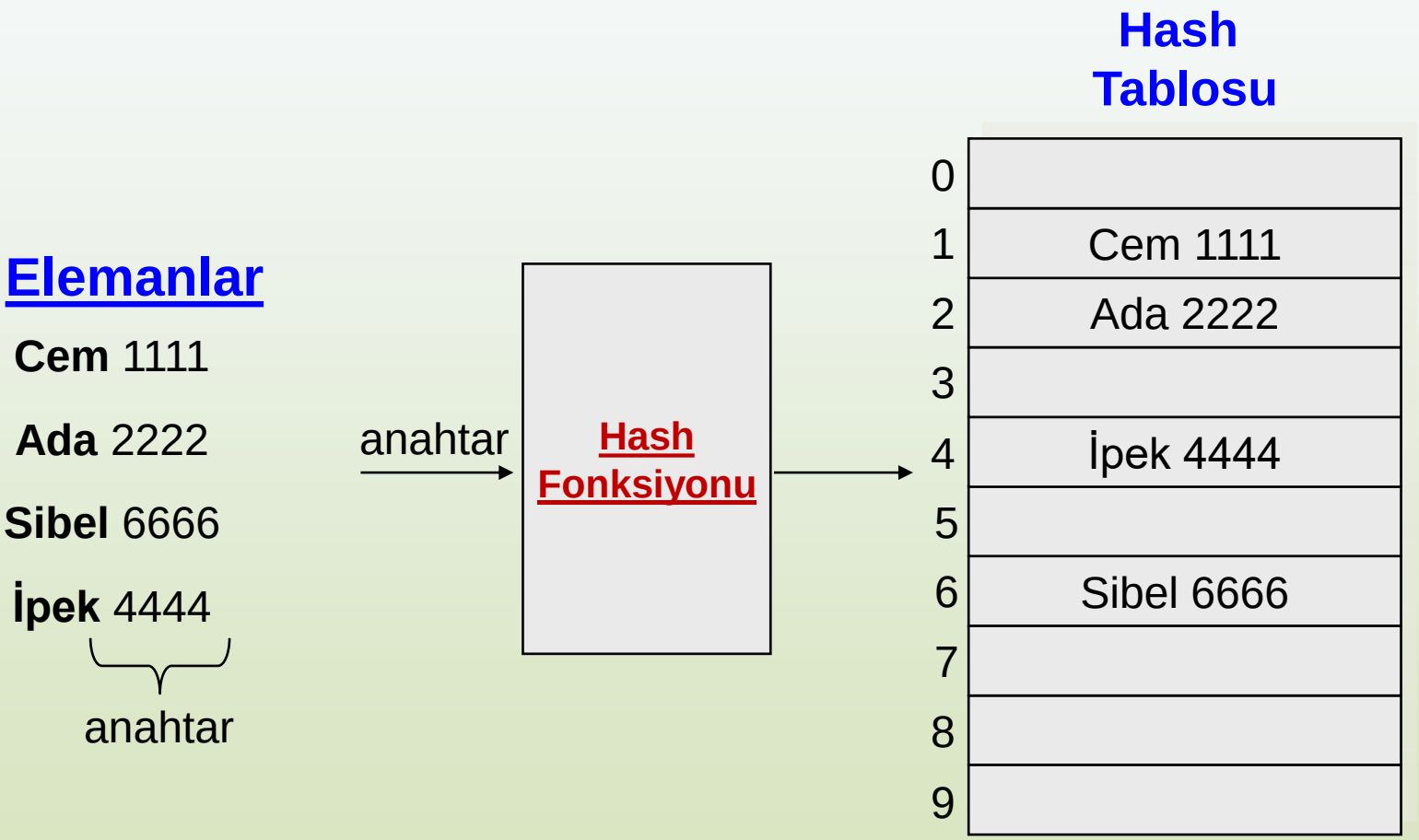
Hash Tabloları

- Hash tablo veri yapısı ile veri arama, ekleme ve silme işlemleri **ortalama olarak sabit zamanda** (O(1)), daha verimli bir biçimde gerçekleştirilir.
- Hash tabloları arama, ekleme ve silme işlemlerinde **ağaçlardan daha hızlı oldukları** (ortalamada) gibi programlanmaları da daha kolaydır.
- Dezavantajları
  - Tablo içerisindeki elemanların ***sıralanması***, ***en büyük*** ya da ***en küçük*** elemanın bulunması işlemlerinde verimli değildir.
  - Aslında böyle bir işlemede yapı itibarı ile gerek yoktur!

Hash Tabloları (devam…)

- İdeal bir hash tablo veri yapısı, içerisinde elemanlar barındıran **sabit bir diziden oluşur**.
- Dizi içerisindeki elemanlar, index hesaplamasında kullanılacak **anahtar (key)** isimli özel bir üye bulundurmak **zorundadırlar**.
  - Anahtar; integer veya **string** bir değer olabilir.
  - **Örneğin:** Bir Öğrenci nesnesindeki Öğrenci No veya Öğrenci TC Kimlik No olabilir.
- Dizinin boyutu ***TabloBoyutu*** olup,
- Diziye eklenecek elemanlar 0'dan ***(TabloBoyutu-1)***'e kadar olan indekslerde saklanırlar.
- **Anahtardan→Indekse** dönüştürme işlemine **Hashing**, bu işi yapan fonksiyona ***Hashing fonksiyonu*** denir.

Örnek


Hash Tablosu
Elemanlar
Cem 1111
Ada 2222
Sibel 6666
İpek 4444
anahtar
anahtar
Hash Fonksiyonu
0
1
Cem 1111
2
Ada 2222
3
4
İpek 4444
5
6
Sibel 6666
7
8
9

Hash Fonksiyonu

- Anahtarın **dizideki pozisyonunu** yani **indisini** belirlemek için kullanılır.
- 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:

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

Hash Fonksiyonu (devam…)

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???**

Hash Fonksiyonu (devam…)

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.

Çakışma (collision) oldu…

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.
- **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)**

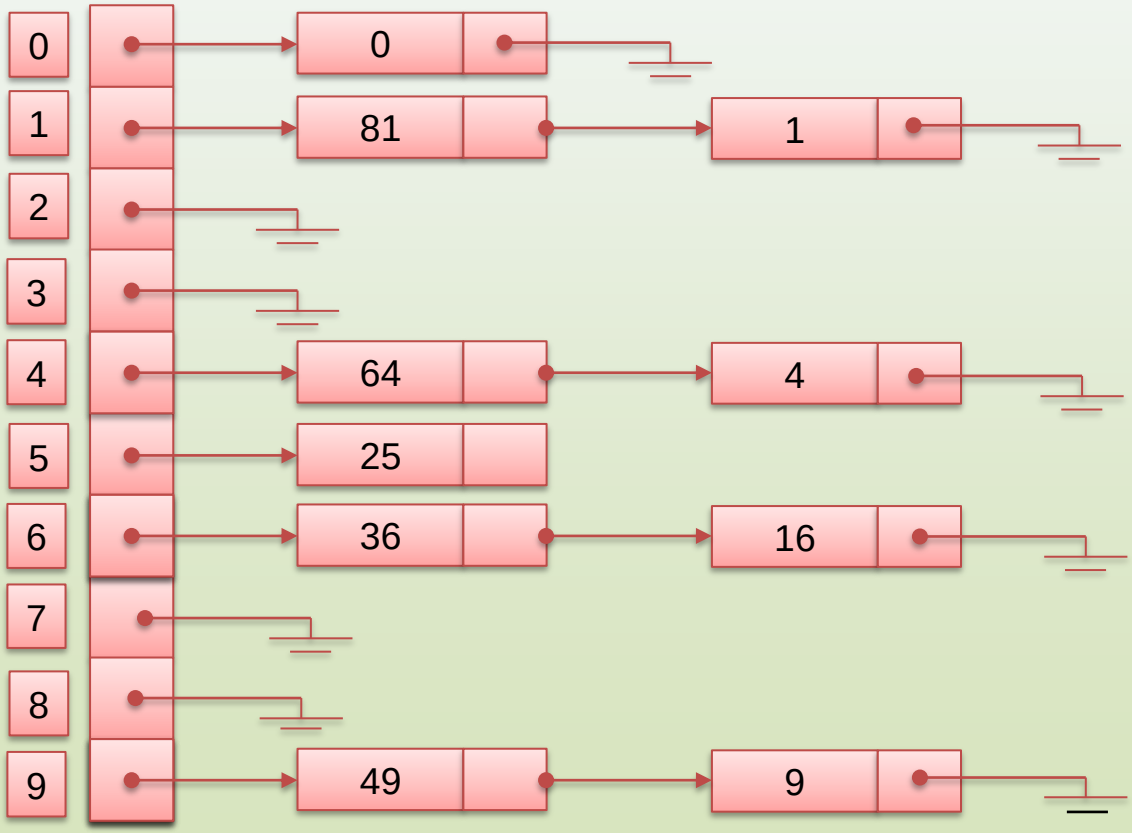
1.Ayrık Zincirleme Çözümü

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

1.Ayrık Zincirleme Çözümü (devam…)

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

hash(anahtar) = anahtar % 10



1.Ayrık Zincirleme Çözümü

- **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.

2. Açık Adresleme Çözümü

- Açık adresleme çözümünde tüm elemanlar **aynı hash tablosunda (dizide)** saklanırlar.
- Çarpışma 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 is the **çakışma (Collision) çözüm stratejisidir**.

2. Açık Adresleme Çözümü

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

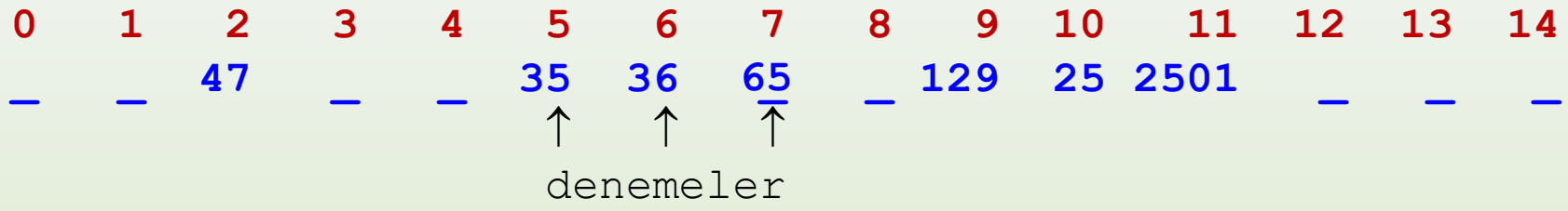
Doğrusal Ölçüm (Linear Probing) (devam…)

- *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

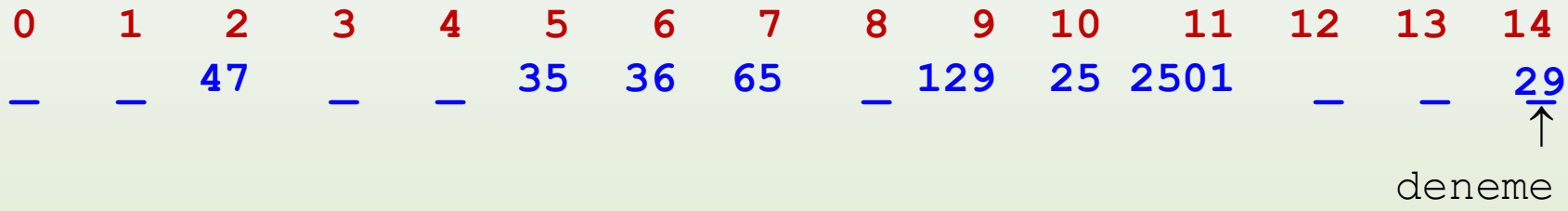


- Toplam **3 deneme** yaptık ve uygun yeri bulduk.

Soru: 29'u nereye ekleyeceğiz?

Doğrusal Ölçüm (Linear Probing) (devam…)

- **Çözüm:** 29 Ekle

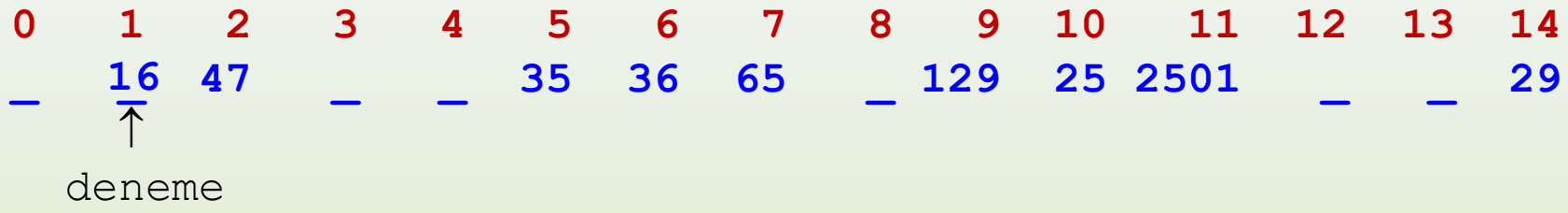


- Toplam **1 deneme** yaptık ve uygun yeri bulduk.

Soru: 16'yı nereye ekleyeceğiz?

Doğrusal Ölçüm (Linear Probing) (devam…)

- **Çözüm:** 16 Ekle

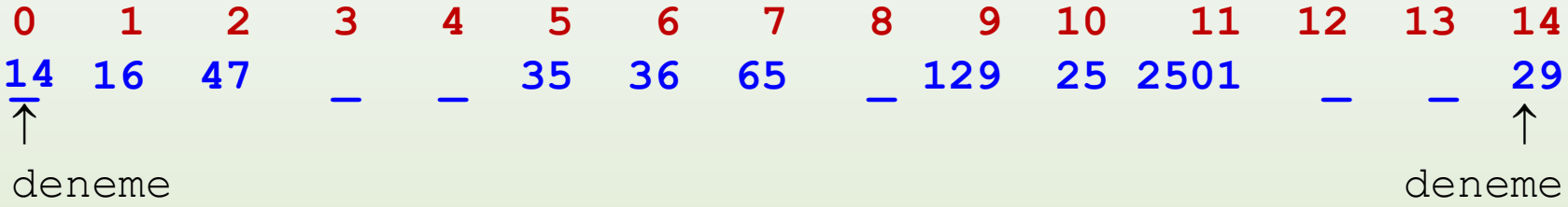


- Toplam **1 deneme** yaptık ve uygun yeri bulduk.

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: Tablonun kabul etmeyeceği 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)***
  - ***Silindi (Deleted)***
- Arama ve silme işlemlerinde
  - Sadece (**durum = Boş**) ise işlemi sonlandır. Silindi durumunda **sonraki elemanla devam et**.

Doğrusal Ölçüm (Linear Probing) (devam…)

- **Ekle**
  - Hücre: Boş veya Silindi — H indise Ekle VE *hücre = Aktif*
  - Hücre: Aktif — H = (H + 1) mod N

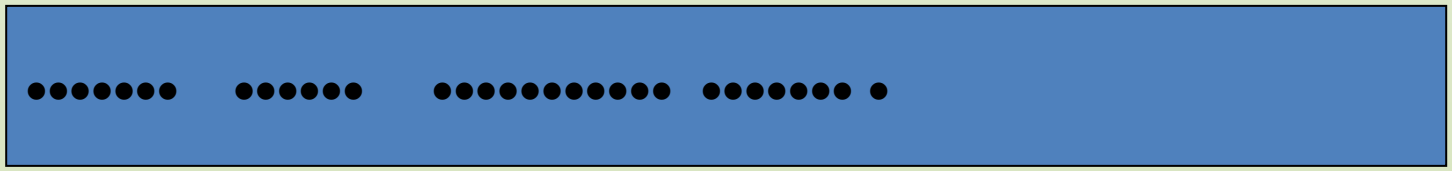
- **Bul / Ara**
  - Hücre: Boş — BULUNAMADI
  - Hücre: Silindi — H = (H + 1) mod N
  - Hücre: Aktif — if key == key in hucre -> BULDU
else H = (H + 1) mod N

- **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

Doğrusal Ölçüm (Linear Probing) (devam…)

Kümeleme (Clustering) Problemi

- Hash tablosu yeterince büyük olduğunda mutlaka boş bir hücre bulunabilir.
- Kayıtların yığın şeklinde toplanmasına yani kümelemeye (clustering) neden olabilir.
- Arama işlemi genelde küme içerisinde gerçekleşir ve kümeye eklenir.

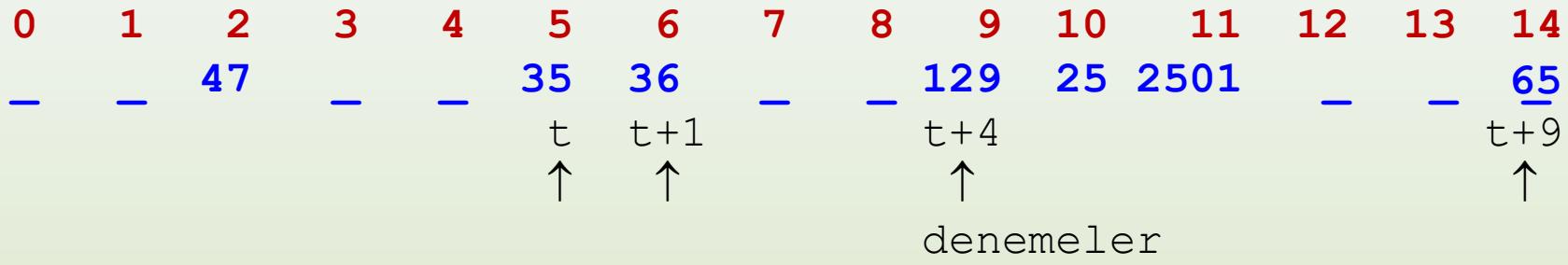


Karesel Ölçüm (Quadratic Probing)

- 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

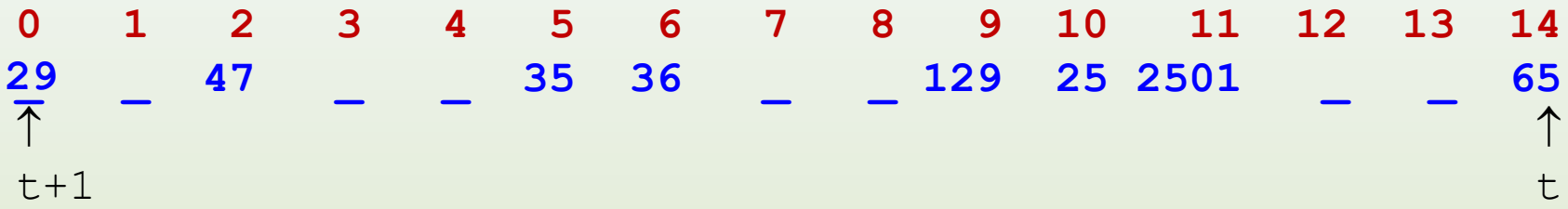


- Toplam **4 deneme** yaptık ve uygun yeri bulduk.

Soru: 29'u nereye ekleyeceğiz?

Karesel Ölçüm (Quadratic Probing) (devam…)

- **Çözüm:** 29 Ekle



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

Hash Fonksiyonları ve Güvenlik

- Hash fonksiyonların güvenlik ve şifreleme alanında da sıkça kullanılmaktadır.
- Hash fonksiyonları tek yönlü (One Way) çalışır. Yani algoritmanın ürettiği sonuçtan **tekrar asıl metine dönüşüm** mümkün değildir.
- Bazı örnek hash fonksiyonları:
  - MD5 (Message Digest 5)
  - MD6 (Message Digest 6)
  - SHA-1 (Secure Hashing Algorithm)
  - HAVAL

Hash Fonksiyonları ve Güvenlik (devam…)

Veritabanında Kullanıcı Bilgisi Saklanması

- Örneğin web sitelerinde kullanıcı bilgilerini veritabanında saklarken SHA kullanılabilir. Web siteleri sizin şifrenizi tek yönlü şifreli tutmak zorundadır.
- 160 bitlik SHA algoritması en çok kullanılanıdır.
- SHA'nın birçok türevi vardır.
- Facebook'un kullandığı 384 bitlik versiyon, 160 bitliğe göre daha güvenlidir.

Hash Fonksiyonları ve Güvenlik (devam…)

Dosya İndirme Kontrolü

- Hash algoritmaların bir diğer kullanım yeri de internetten indirdiğiniz herhangi bir dosyanın tam inip inmediğini tespit etmektir.
- Örneğin, web sitemizde 700 MB'lık bir Ubuntu.ISO dosyasının linkini koyduk.
- Linkin altına Ubuntu.ISO dosyasının SHA veya MD5 algoritmasından geçirilmiş halini de koyuyoruz.
- Ubuntu.ISO dosyasını sitemizden indiren kullanıcı, kendi bilgisayarında indirdiği dosyayı MD5 veya SHA'dan geçiriyor, sitemizdeki ile aynı sonuç çıkarsa, sorunsuz inmiştir.

İYİ ÇALIŞMALAR…

Yararlanılan Kaynaklar

- **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.