



Veri Yapıları ve Programlama

2

3.HAFTA

Listeler

BÖLÜM - 3

Bu bölümde,

- **Motivasyon:** Neden Listeye İhtiyaç Var?
- Bağlı Liste (Linked List) Veri Yapısı

konusuna değinilecektir.

Motivasyon: Neden Listeye İhtiyaç Var?

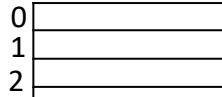
4

- **Dizi**, bellek yöneticisi tarafından atanan ve uzunluğu baştan belli ardışık bir hafızada saklanır.
- `int[] A = {6, 5, 4, 2}` şeklinde 4 elemanlı **int** türünde bir dizi olduğunu varsayalım.
- Hafızaya sırayla
 - `int x = 8;` sayısını ve
 - daha sonra A dizisinisakladığımızı düşünelim.

Motivasyon: Neden Listeye İhtiyaç Var?

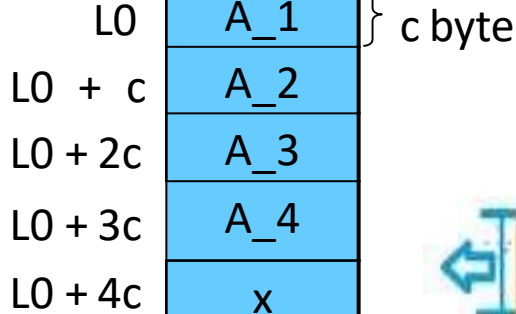
5

Hafıza

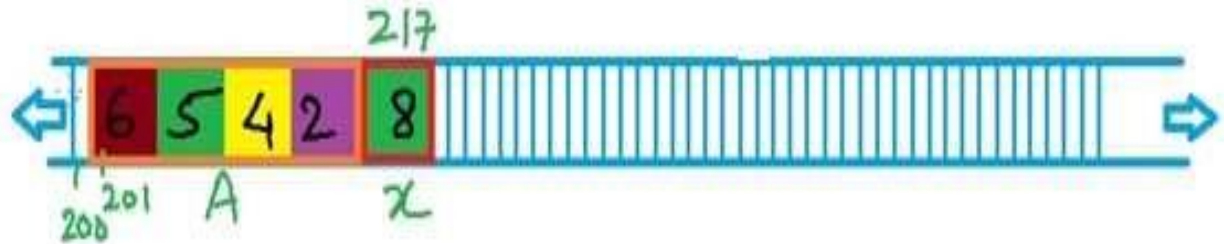


```
int x = 8;  
int[] A = {6, 5, 4, 2};
```

Adres



Nasıl bir bellek organizasyonu bekliyoruz?



MAUZEM

Motivasyon: Neden Listeye İhtiyaç Var?

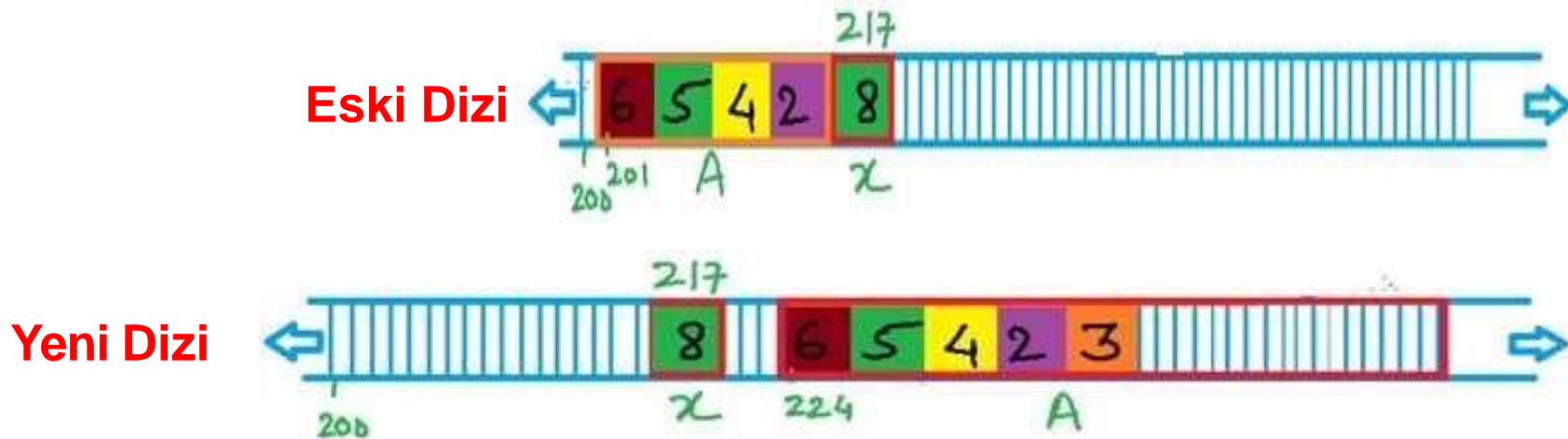
6

- Dizi için ayrılan bölümün sürekli adreslerden oluştuğuna dikkat edelim.
- Bu diziye **yeni bir eleman** (örneğin 3) eklemek istediğimizde
 - 2'den sonraki alan, yani 217 nolu hafıza dolu olduğu için hafızayı genisletme seçeneğimiz yoktur.
- **Çözüm olarak ne yapabiliriz?**

Motivasyon: Neden Listeye İhtiyaç Var?

7

- Çözüm olarak birçok **hafıza değişimi ve işlemi gerektiren** aşağıdaki adımlar uygulanır:
 - *Daha büyük bir dizi* için yer alanı oluşturulur.
 - Eski dizideki elemanlar yeni diziye **kopyalanır** ve **taşım** (move) işlemi gerçekleştirilir.
 - **Eski dizi** hafızadan **silinir**.



Motivasyon: Neden Listeye İhtiyaç Var?

8

- Bu problemi **çözmek** için ne **yapabiliriz?**

ÇÖZÜM 1: Büyük Hafıza Alanı

- Büyük bir hafıza alanını **dizi için ayırabiliriz.**
- Ancak bu durumda da **hafızada saklanan eleman sayısı azken hafıza boşa işgal edilmiş** olur.

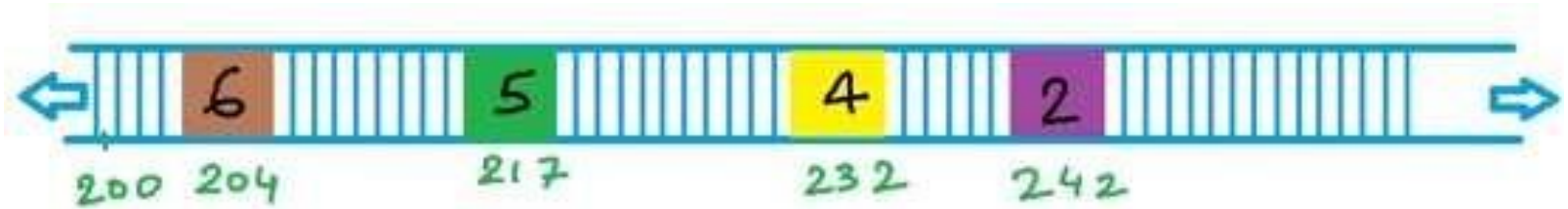
ÇÖZÜM 2: Bağlı Liste

- Hafızayı **ihtiyaç halinde büyütebilmektir.**
- **Önceden alan ayırmayarak** hafızayı *verimli kullanmak adına* elemanları **elemanları hafızaya tek tek yerleştirmektedir.**

Bağlı Liste – Çözüm

9

- `int[] A = {6, 5, 4, 2}` dizisini aşağıdaki gibi tutabildiğimizi varsayalım.

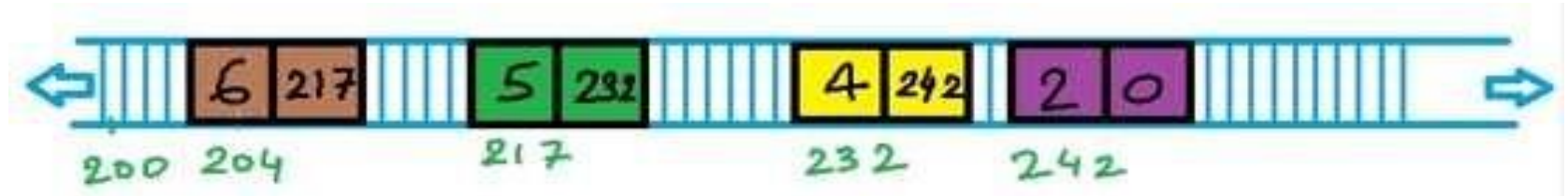


- Elemanlar arasında bir **bağlantı** sağlanması **zorunludur**. **Neden?**

Bağlı Liste – Çözüm (devam...)

10

- Hafıza sürekli olmadığı için dizi elemanlarının birbirleriyle **bağı yoktur**. Bu nedenle her bir eleman *sonrakinin adresini gösteren* bir referansla bağlanmalıdır.

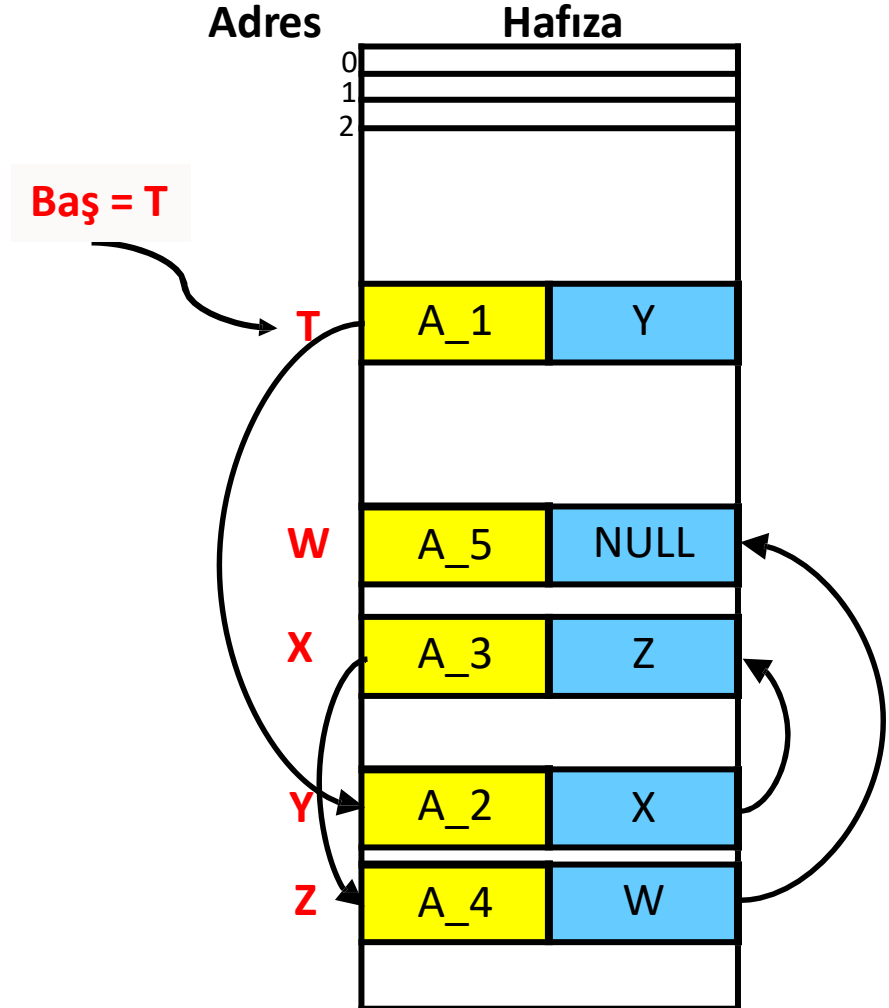


- Her eleman sonraki elemanın adresini tutarken **son elemanın adresi NULL** olur.
- Bu şekilde elde edilen veri yapısı, **Bağlı Liste (Linked List)** olarak adlandırılır.

Bağlı Liste – Çözüm (devam...)

11

- Bağlı listenin hafıza organizasyonu ve yerleşimi yandaki gibi de olabilir.

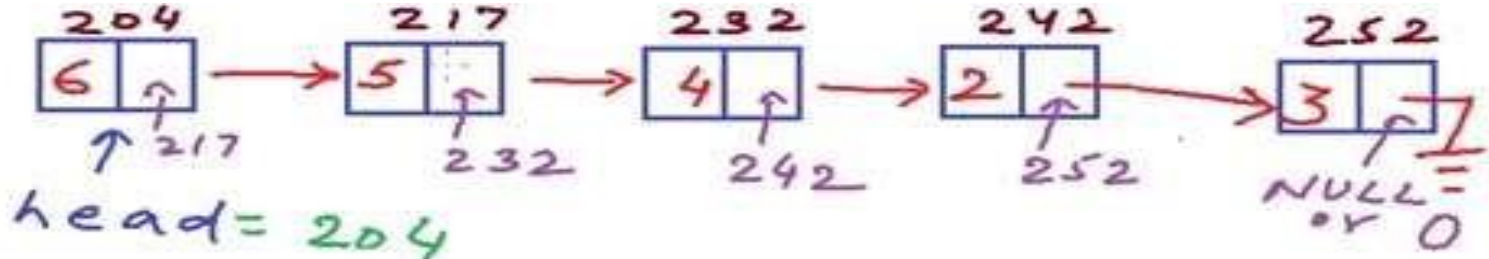


Bağlı Liste – Çözüm (devam...)

12

MAUZEM

- Listenin sonuna “3” sayısını eklersek,
- Yeni eleman NULL'a işaret ederken, yani listenin son elemanı olurken,
- 2 artık “252” ye (**3'ün adresine**) işaret eder.
- Aşağıdaki şekilde dizinin bağlı liste yapısı verilmiştir.



Bağlı Liste – Özet

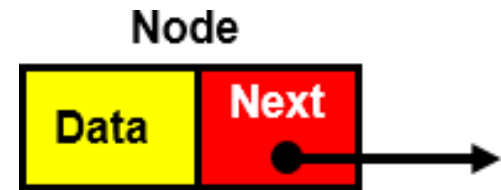
13

- Bir bağlı liste, elemanlarının **bir sonraki (Next)** elemanın **hafızadaki yerine** işaret ettiği zincir **şeklinde** bir veri yapısıdır.
- 3 tip bağlı liste bulunmaktadır.
 - Tek yönlü bağlı liste
 - Çift yönlü bağlı liste
 - Dairesel bağlı liste

Bağlı Liste – Özet (devam...)

14

- Bağlı listede her bir **elemana düğüm (node)** adı verilir.
- Her **düğüm**
 - **Data**: Veri (bir tamsayı, bir dizi, bir nesne olabilir)
 - **Next**: *Sonraki verinin adresine* işaret eden bir referanstan oluşur.

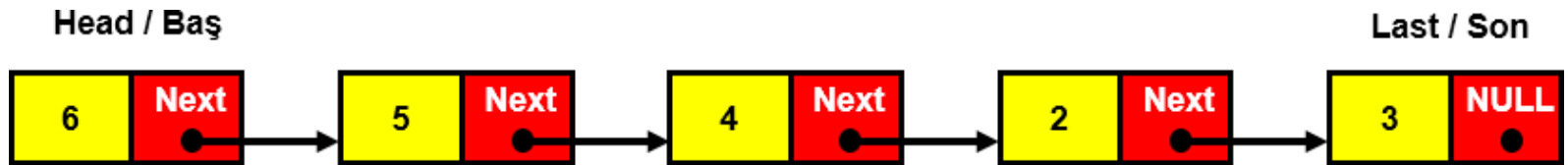


```
public class Node
{
    public int Data;
    public Node Next;
}
```

Bağlı Liste – Özet (devam...)

15

- **İlk düğümün** adresini içeren ve **başlangıç (Head)** olarak adlandırılan bir *referans* değişkeni bulunur.
 - Bir listeyi **gezmek** için bu adrese mutlaka ihtiyaç vardır.
 - Boş bir listede **Head NULL**'a işaret eder.
- Bağlı listede, **son düğümde listenin sonuna işaret eden özel NULL** (veya sıfır) değeri bulunur.



Bağlı Liste – ADT

16

```
public abstract class LinkedListADT
{
    public Node Head;
    public int Size = 0;
    public abstract void InsertFirst(int value);
    public abstract void InsertLast(int value);
    public abstract void InsertPos(int position, int value);
    public abstract void DeleteFirst();
    public abstract void DeleteLast();
    public abstract void DeletePos(int position);
    public abstract Node GetElement(int position);
    public override string DisplayElements()
}
public class Node
{
    public int Data;
    public Node Next;
}
```


Bağlı Liste – ADT (devam...)

17

MAUZEM

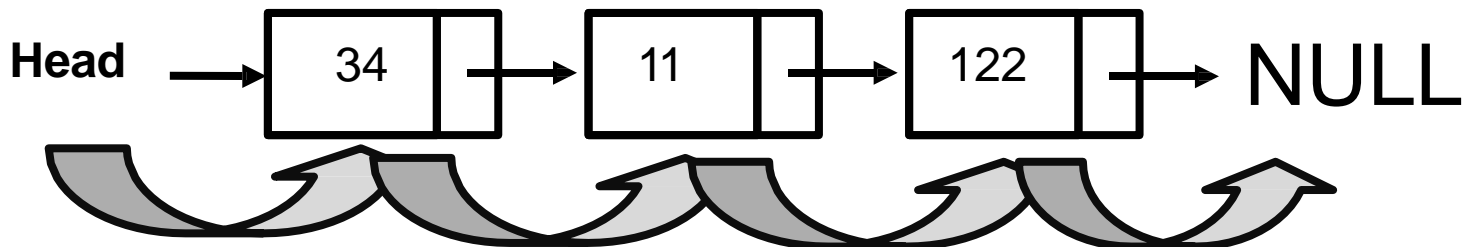
- InsertFirst(int value): Listenin başına bir node ekler.
- InsertLast(int value): Listenin sonuna bir node ekler.
- InsertPos(int position, int value): Listenin belirtilen pozisyonuna bir node ekler.
- DeleteFirst(): Listenin başındaki node'u siler.
- DeleteLast(): Listenin sonundaki node'u siler.
- DeletePos(int position): Listenin belirtilen pozisyondaki node'unu siler.
- GetElement(int position): Listenin belirtilen pozisyondaki node'unu getirir.
- int Size: Listedeki eleman sayısını döndürür.
- string DisplayElements(): Listedeki elemanları geriye döndürür.
- Node Head: Listenin ilk elemanını verir.

Bağlı Liste – Traverse (Gezinme)

18

- Head düğümünden başlayarak, **sonuncu düğüme kadar** doğrusal şekilde **nasıl ilerleriz?** Algoritma, mantık?

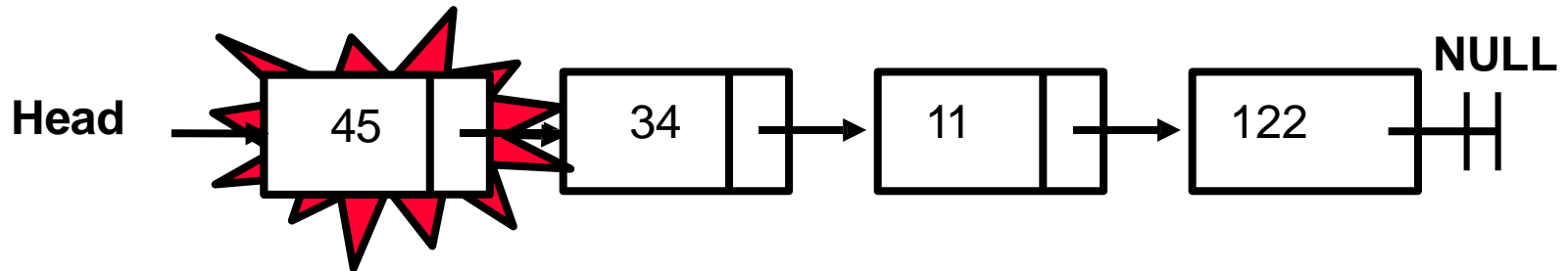
```
□ Node temp = Head;  
□ while (temp != null)  
□ {  
    □ if (temp.Next != null)  
      temp = temp.Next;  
    □ else  
      □ break;  
□ }
```



Bağlı Liste – InsertFirst

19

- Listenin ilk elemanı olan Head'i yeni gelen elemanla yer değiştirdiğimiz metottur.
- **Traverse** işlemine **gerek yoktur**.
- Aşağıdaki listede *Head 34*'tür.
- Listenin başına 45 elemanını eklemek istiyoruz.
Yani yeni Head 45 olmalı. **Nasıl?**



Bağlı Liste – InsertFirst (devam...)

20

- **Sözde kod:**

- a) Yeni Head düğüm oluşturulur (**tmpHead**),
- b) **Head NULL** ise (**Liste BOŞSA**)
 - **Head** \leftarrow **tmpHead**
- c) Yeni düğümün işaretçisi **Head'a işaret eder**,
 - **tmpHead.Next** \leftarrow **Head**
- d) **Head yeni düğüme işaret eder.**
 - **Head** \leftarrow **tmpHead**

Bağlı Liste – InsertFirst (devam...)

21

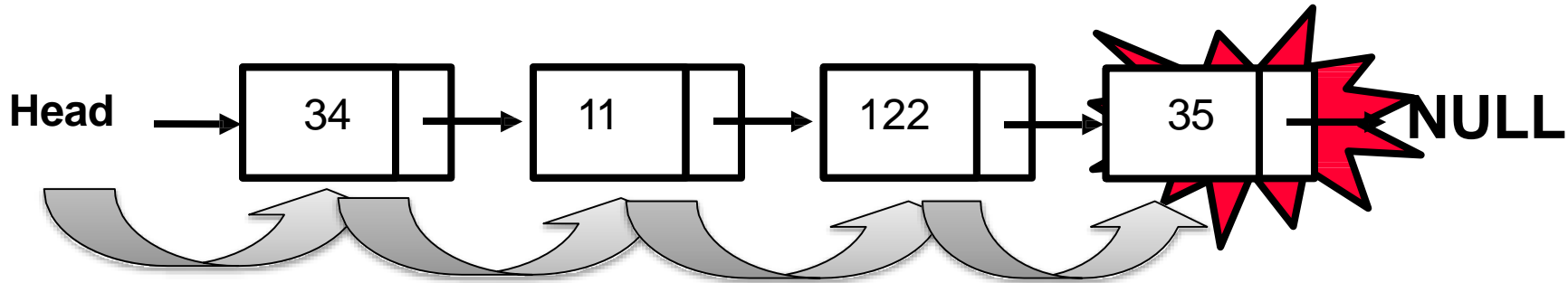
```
public override void InsertFirst(int value)
{
    Node tmpHead = new Node
    {
        Data = value
    };

    if (Head == null)
        Head = tmpHead;
    else
    {
        //En kritik nokta: tmpHead'in next'i
        //eski Head'i göstermeli
        tmpHead.Next = Head;
        //Yeni Head artık tmpHead oldu
        Head = tmpHead;
    }
    //Bağlı listedeki eleman sayısı bir arttı
    Size++;
}
```

Bağlı Liste – InsertLast

22

- Listenin son elemanını yeni gelen elemanla yer **değiştirdiğimiz** metottur.
- **Traverse** işlemi **gereklidir**. Eski sonuncu elemanın *bulunması* gerekmektedir.
- Aşağıdaki listede *Son eleman 122'dir*.
- Listenin sonuna 35 elemanını eklemek istiyoruz.
Nasıl?



Bağlı Liste – InsertLast (devam...)

23

- **Sözde kod:**

a) Yeni düğüm oluşturulur (**newLast**),

b) **Head NULL** ise (**Liste BOŞSA**)

- **InsertFirst()**

c) **“Eski son düğüm”** bulunur (**oldLast**),

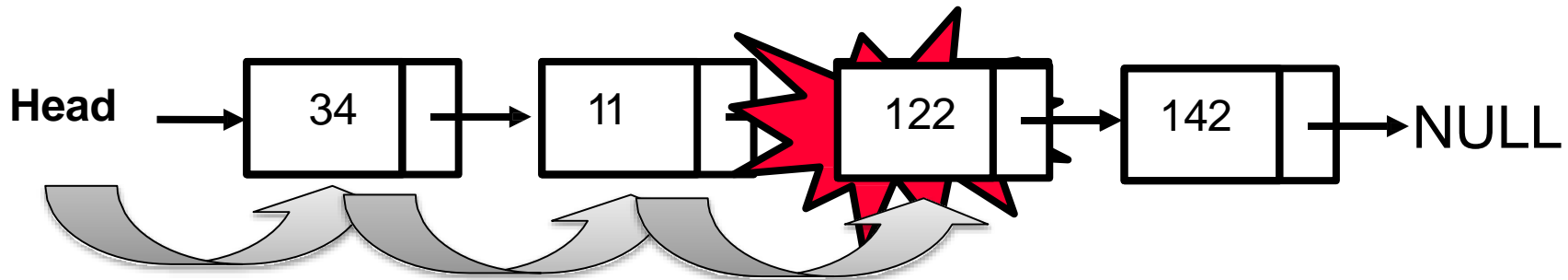
d) **Eski son düğüm**, eklenen düğüme *işaret eder*.

- **oldLast.Next ← newLast**

Bağlı Liste – InsertPos

24

- Listenin **i. pozisyonundaki** elemanın sonrasına **yeni gelen elemanı, ekleyen** metottur.
- **Traverse** işlemi **gereklidir**. **i. Pozisyona sahip elemanın bulunması** gerekmektedir.
- Aşağıdaki listede ***i=2 için 11 elemanı ile 142 elemanı arasında 122 elemanını ekleyelim. Nasıl?***



Bağlı Liste – InsertPos (devam...)

25

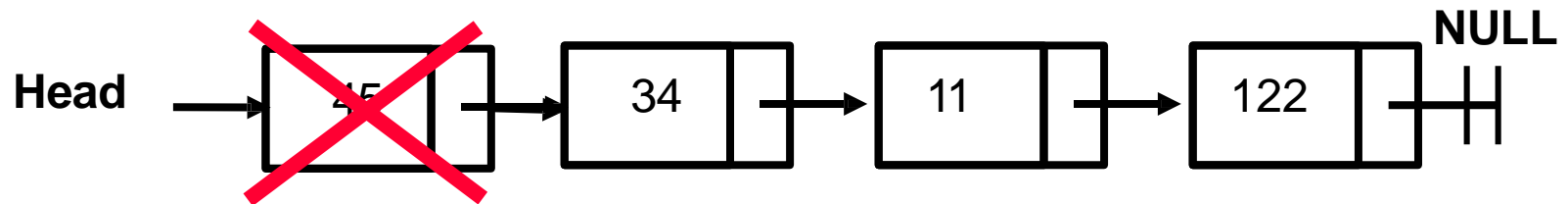
- **Sözde kod:**

- a) Yeni düğüm oluşturulur (**newNode**),
- b) **Head** **NULL** ise (**Liste BOŞSA**)
 - **InsertFirst()**
- c) **“i. Pos düğüm”** bulunur (**posNode**),
- d) **posNode** elemanının Next'i **tempNext'e** atanır.
 - **tempNext ← posNode.Next**
- e) **posNode** elemanının Next'ine **newNode** atanır.
 - **posNode.Next ← newNode**
- f) **newNode** elemanının Next'ine **tempNext** atanır.
 - **newNode.Next ← tempNext**

Bağlı Liste – DeleteFirst

26

- Listenin ilk elemanı olan Head'i **silen** metottur.
- **Traverse** işlemine **gerek yoktur**.
- Aşağıdaki listede *Head 45'dir*.
- Listenin başındaki elemanı silmek istiyoruz.
- Yeni Head **34 olmalı**. **Nasıl?**



Bağlı Liste – DeleteFirst (devam...)

27

- **Sözde kod:**

a) Head'in sonraki elemanı - Next'i temp bir değişkene atanır

- `tempHeadNext ← Head.Next`

b) `tempHeadNext NULL ise` Head NULL olur,

- Liste boşalır

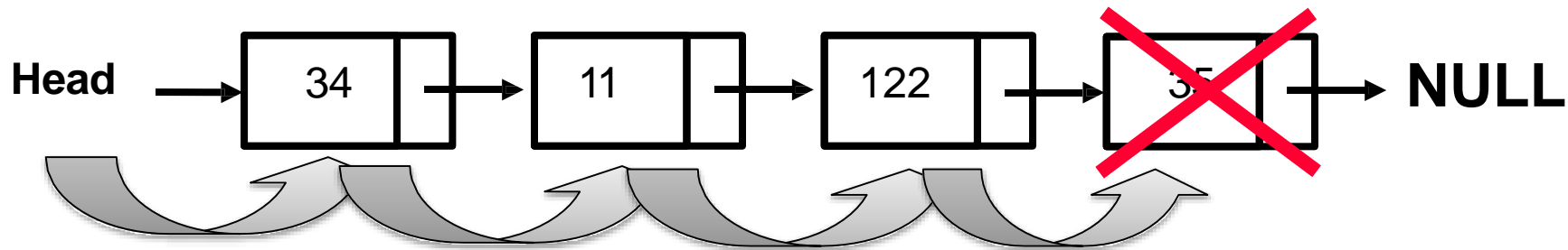
c) `tempHeadNext NULL değilse` Head elemanı `tempHeadNext` olur,

- `Head ← tempHeadNext`

Bağlı Liste – DeleteLast

28

- Listenin son elemanını **silen** metottur.
- **Traverse** işlemi **gereklidir**. **Sonuncu elemanın**
 - *bulunması* gerekmektedir.
- Aşağıdaki listede *Son eleman 35'dir*.
- Son eleman silindiğinde **yeni son eleman 122**
 - **olmalıdır**. **Nasıl?**



Bağlı Liste – DeleteLast (devam...)

29

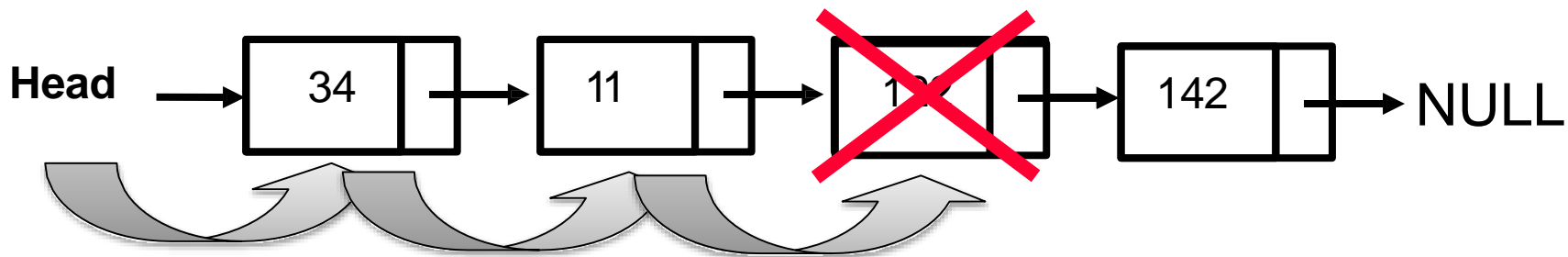
- **Sözde kod:**

- a) Son eleman bulunması için Head'ten itibaren doğrusal ilerlenir.
- b) Son eleman **last** değişkenine atanır.
- c) Son eleman bulunduğunda, **sondan bir önceki eleman** bulunur ve bir değişkene atanır (**lastPrevNode**).
- d) **lastPrevNode** elemanının **Next**'ine **NULL** **atanır**.
- e) **last** değişkenine **NULL** atanır.

Bağlı Liste – DeletePos

30

- Listenin **i. pozisyonundaki** elemanını **silen** metottur.
- **Traverse** işlemi **gereklidir**. **i. Pozisyona sahip elemanın bulunması** gerekmektedir.
- Aşağıdaki listede **$i=2$ için 122 elemanını silelim ve 11 elemanı artık 122'yi değil, 142'yi işaret etsin. Nasıl?**

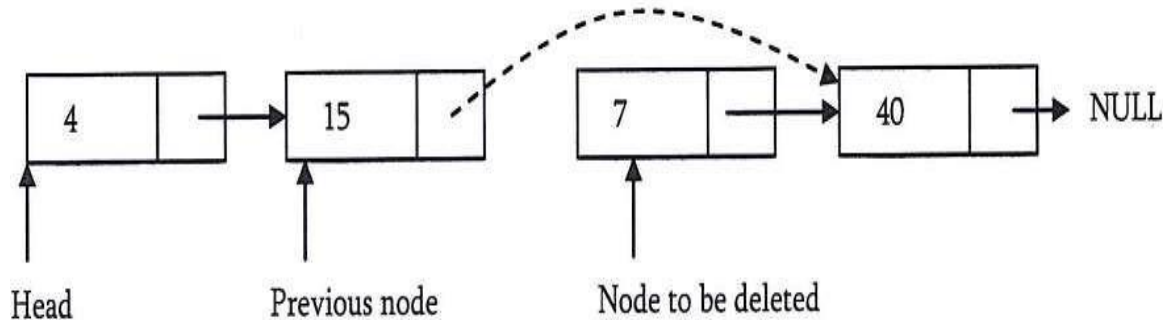


Bağlı Liste – DeletePos (devam...)

31

- **Sözde kod:**

- a) *i*. Pozisyondaki elemanın bulunması için Head'ten itibaren doğrusal ilerlenir.
- b) *i*. Pozisyonundaki eleman **pos** değişkenine atanır.
- c) *i*. Pozisyonundaki eleman bulunduğunda, **ondan bir önceki elemanın Next'ine pos'un Next'i atanır.**
- d) **pos** değişkenine **NULL** atanır.



Bağlı Liste ve Dizi Karşılaştırması

32

- **Erişim Maliyeti**

- Dizinin 1. veya en sonuncu elemanına erişmek için sadece base adresle **ilgili elemanın sırasının** **byte değeri** (6. sıra * 4 byte) toplanır. Bu her eleman için sabit bir zamandır ve karmaşıklığı **$O(1)$** olur.
- Bağlı liste, **sürekli bir adreste tutulmadığından** en kötü durumda (**worst-case**) **son elemanın adresine erişmek** için Head elemanından başlanarak **sırayla** her düğümden *bir sonraki adres elde* edilir. Bu nedenle karmaşıklık eleman sayısı n kadar ya da **$O(n)$** olur.

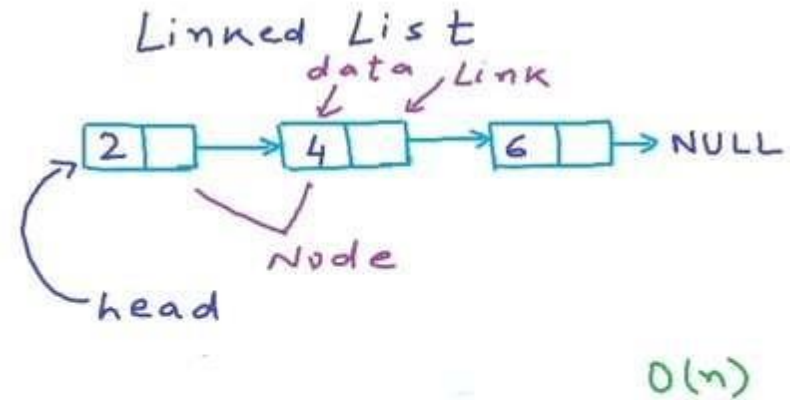
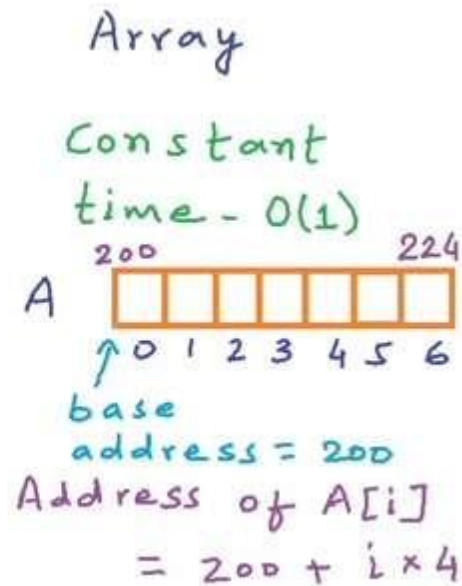
Bağlı Liste ve Dizi Karşılaştırması

(devam..)

33

- **Erişim Maliyeti**

1) Cost of accessing an element



Bağlı Liste ve Dizi Karşılaştırması

(devam..)

34

- **Veri Giriş (Insert) Maliyeti**
 - **En Başa:** Dizide tüm elemanlar bir kayar $O(n)$. Bağlı listede sabit bir işlem $O(1)$.
 - **En Sona:** Dizi boşsa $O(1)$ değilse yeni bir dizi gerekeceğinden en kötü $O(n)$. Bağlı listede sonuncu elemana en kötü $O(n)$ le ulaşılır ve eklenir.
 - **i. Pozisyona:** Dizide ilgili elemana ulaşmak ve duruma göre elemanları kaydırmak gerekeceği için en kötü durumda $O(n)$ zamana ihtiyaç vardır. Bağlı listede ise kaydırma olmadığı halde bir elemana erişmek için head adresinden istenen adrese kadar gezmek gerekeceğinden en kötü durumda $O(n)$ olur.

Bağlı Liste ve Dizi Karşılaştırması

(devam..)

- Veri Giriş (Insert) Maliyeti

3) Cost of inserting an element

Array

A

2	4	6	5	-	-	-
0	1	2	3	4	5	6

Linked List

head →

2	→
---	---

 →

4	→
---	---

 →

6	→
---	---

 →

5	→ NULL
---	--------

a) at beginning - $O(n)$ $O(1)$

b) at end - $O(1)$ $O(n)$

c) at i^{th} position - $O(n)$ $O(n)$

If array is not full - $O(n)$ - array full

Bağlı Liste ve Dizi Karşılaştırması

(devam..)

36

- **Silme (Delete) Maliyeti**
 - Insert işlemleri ile tamamen aynı senaryolara ve aynı Big-O karmaşıklık maliyetlerine sahiptirler.

Bağlı Liste ve Dizi Karşılaştırması

(devam..)

37

İşlem	Dizi Tabanlı Liste	Bağlı Liste
Access, Retrieve (Erişim)	$O(1)$	$O(n)$
InsertFirst	$O(n)$	$O(1)$
InsertLast	$O(n)$	$O(n)$ Sonuncu elemana pointer varsa = $O(1)$
InsertPos	$O(n)$	$O(n)$
DeleteFirst	$O(n)$	$O(1)$
DeleteLast	$O(n)$	$O(n)$ Sonuncu elemana pointer varsa = $O(1)$
DeletePos	$O(n)$	$O(n)$

Telif ve Kaynaklar

Doç.Dr. Deniz KILIÇ'ın ders notlarından yararlanılmıştır.

Kaynak gösterilmek şartıyla her türlü kullanıma açıktır.