BMB2006
VERİ YAPILARI

Doç. Dr. Murtaza CİCİOĞLU

Bursa Uludağ Üniversitesi

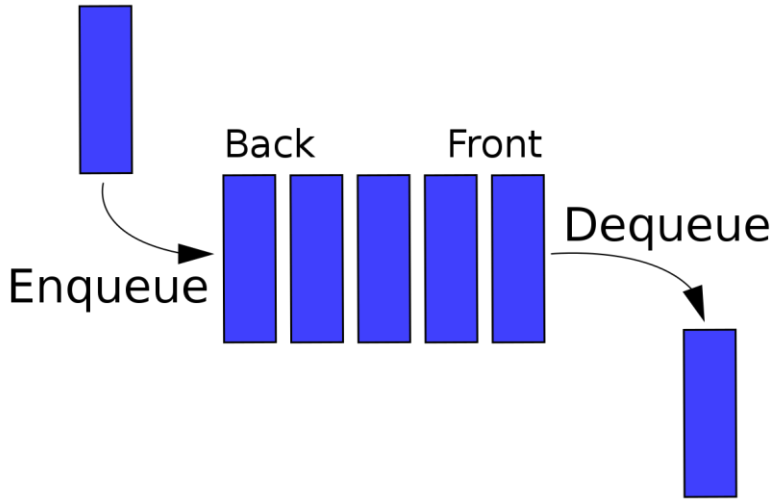
Bilgisayar Mühendisliği Bölümü

Bu ders notları Olcay Taner Yıldız ile Erkan TANYILDIZI hocaların notları kullanılarak hazırlanmıştır.

Hafta 6: Kuyruk (Queue)

Amaç:

- Kuyruk çalışma yapısı
- Kuyruk kullanım alanları

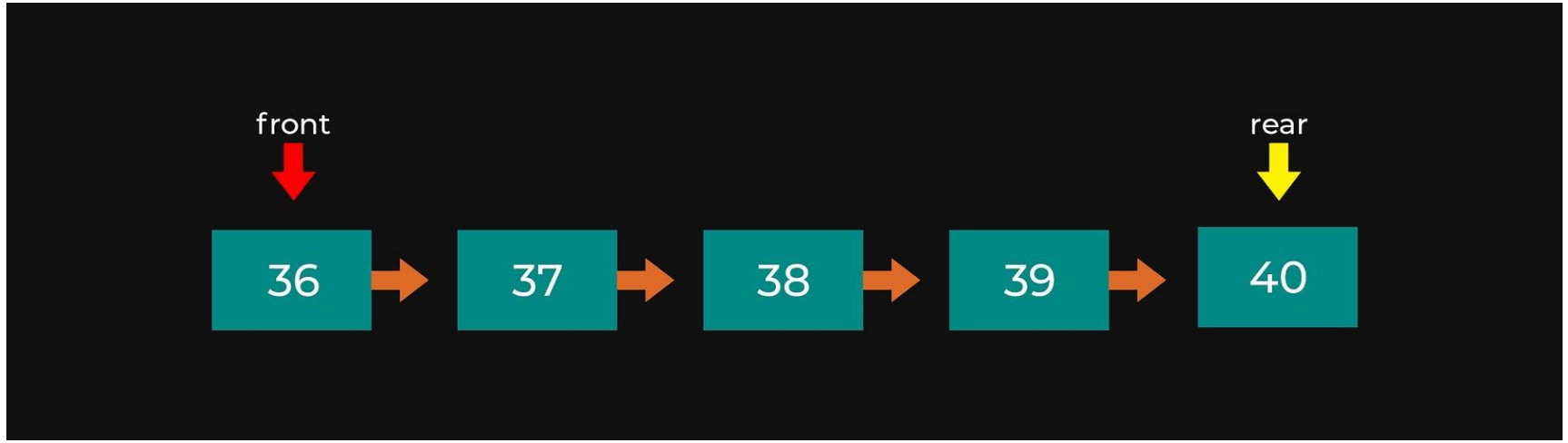


Yol haritası:

- Giriş
- Dizi ile Kuyruk Tanımı
- Bağlı Liste ile Kuyruk Tanımı
- Hedef Tahtası

Giriş

- Kuyruklar, eleman eklemelerinin sondan ve eleman çıkarmalarının baştan yapıldığı doğrusal veri yapılarıdır.
- Bu nedenle kuyruklara FIFO (First In First Out-İlk giren ilk çıkar) veya LILO (Last-in-Last-out-Son giren son çıkar) listeleri denilir.



Giriş

- Queue yapısını gerçekleştirmek için 2 yol vardır.
  - Dizi kullanmak (arttırımsal, dairesel)
  - Bağlantılı liste kullanmak

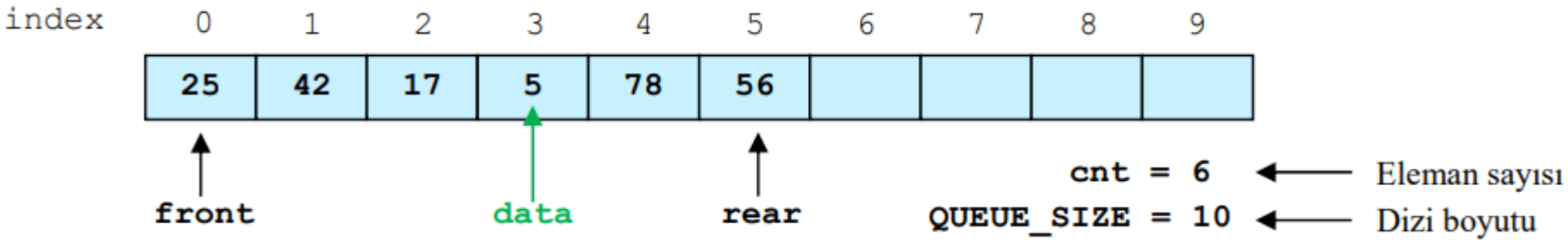
- Stack İşlemleri
  - enqueue(nesne)
  - dequeue()
  - peek()
  - size()
  - isEmpty()

Kullanım alanları

- Kuyruk (queue) yapısı bilgisayar alanında; ağ, işletim sistemleri, istatistiksel hesaplamalar, simülasyon ve çoklu ortam uygulamalarında yaygın olarak kullanılmaktadır.
- Örneğin, yazıcı kuyruk yapısıyla işlemleri gerçekleştirmektedir. Yazdır komutu ile verilen birden fazla belgeden ilki yazdırılmakta, diğerleri ise kuyrukta bekletilmekte, sırası geldiğinde ise yazdırılmaktadır.

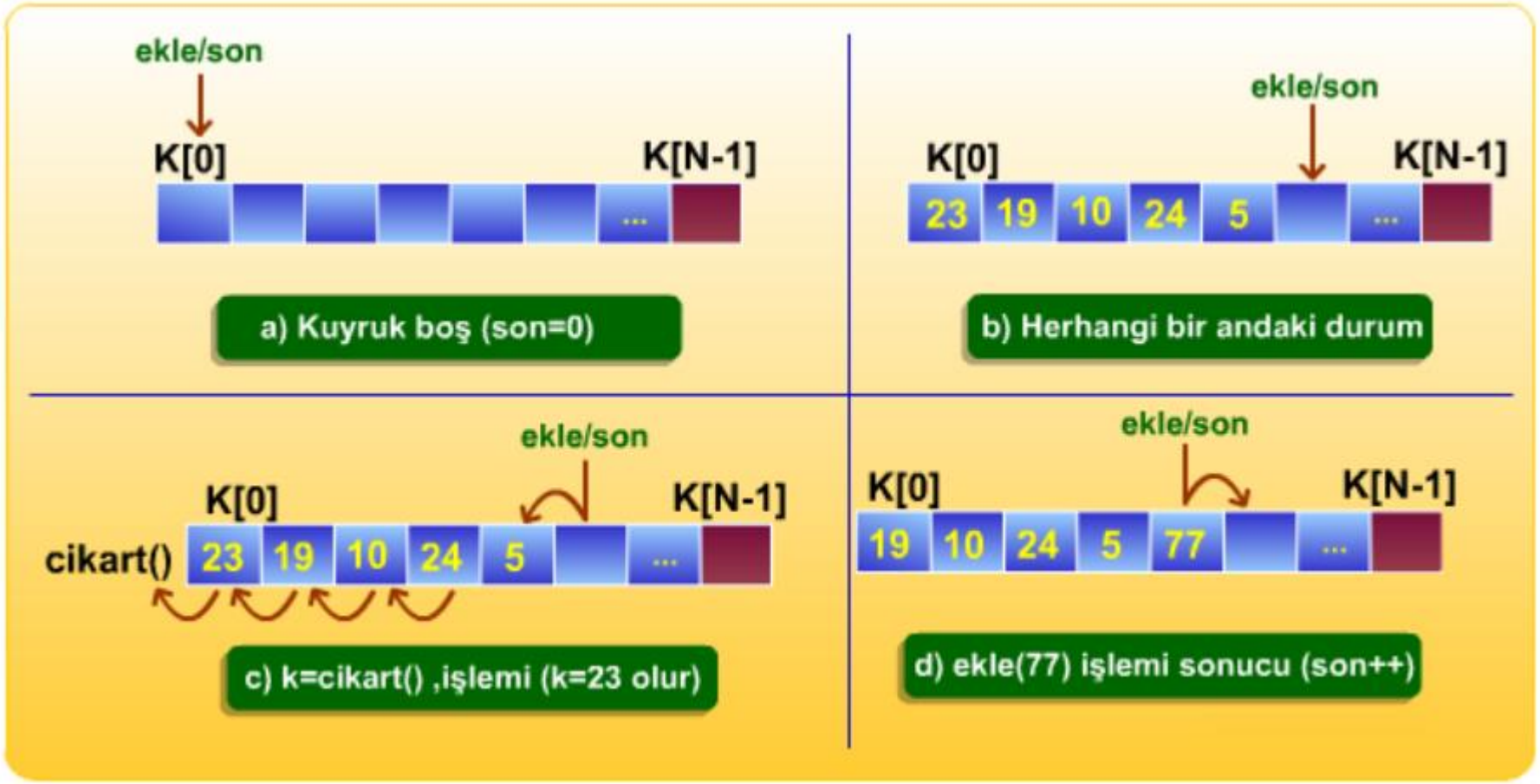
Dizi tabanlı

- Bir queue gerçeklemenin en kolay yolu dizi kullanmaktır. queue yapısı dizi üzerinde en fazla N tane eleman tutacak şekilde yapılabilir.



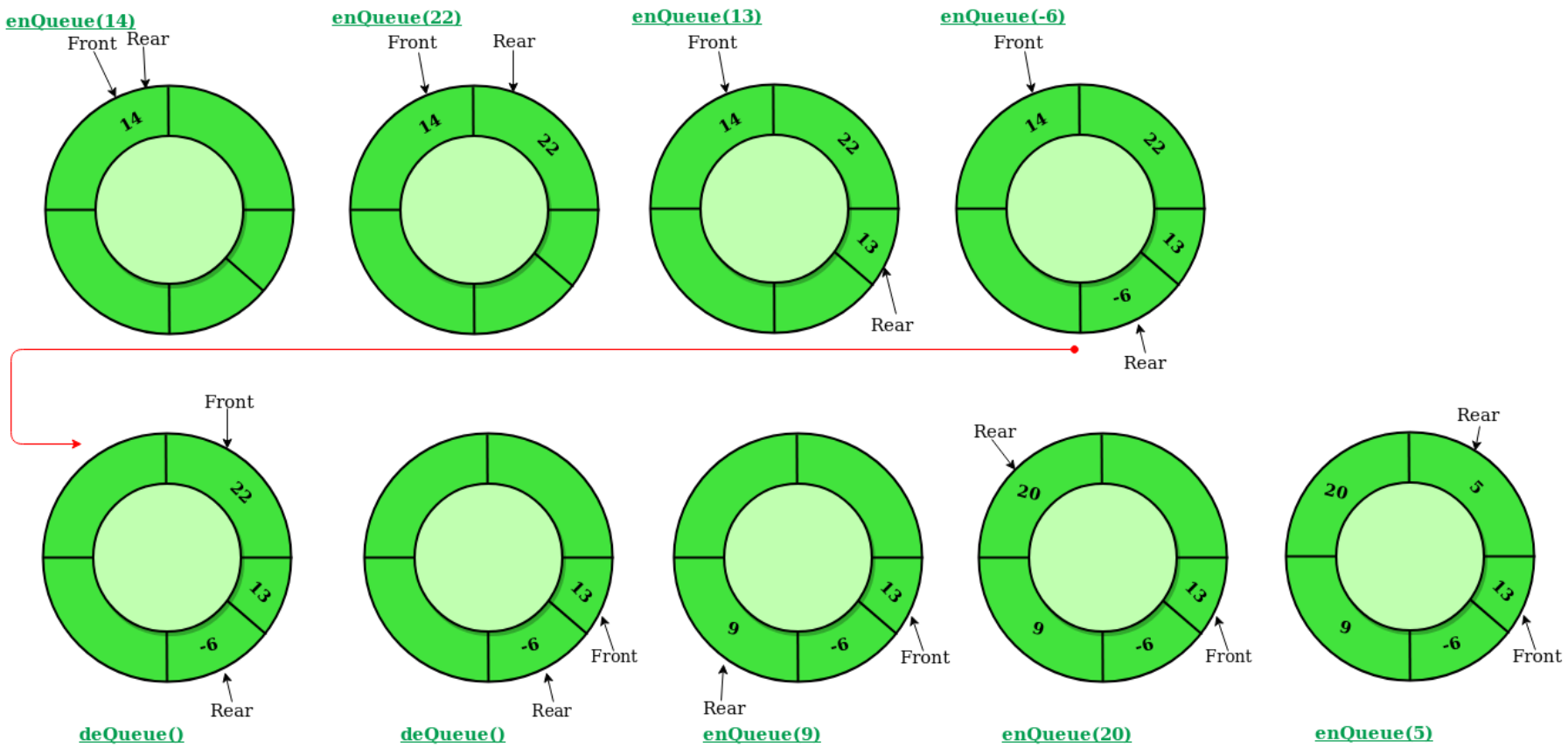
Dizi tabanlı

- N uzunluktaki bir dizi üzerinde kaydırmalı kuyruk yapısının davranışı şekilde



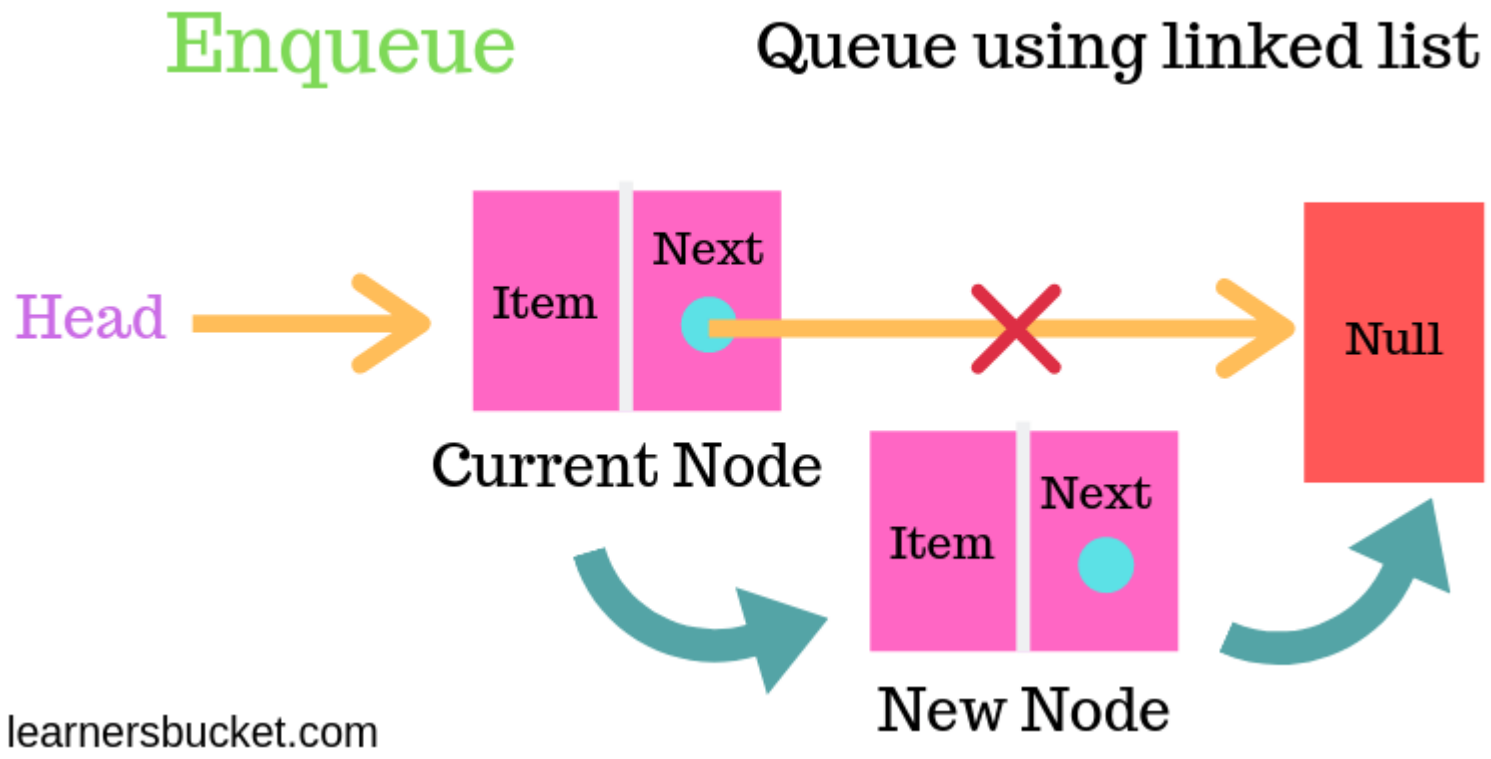
Dizi tabanlı

- Dairesl kuyruk tasarımı



Linked List tabanlı

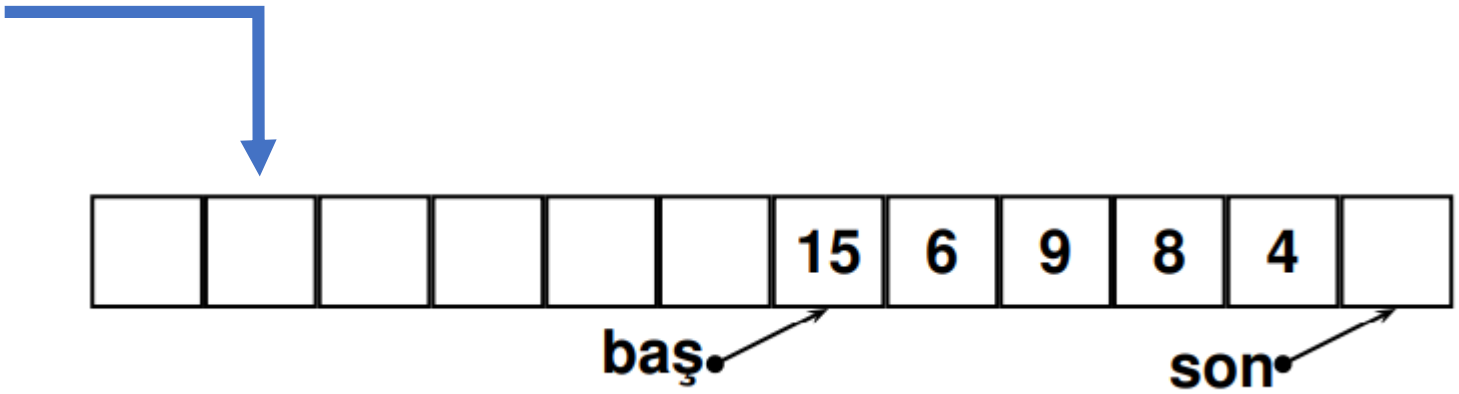
- Bir queue gerçeklemenin bir diğer bağlı liste kullanmaktır.



Dizi tabanlı

Sabit dizi ile tanımlı bir kuyruk
ait elemanların tanımı

```
public class Eleman{
        public int veri;
        public Eleman(int veri){
            this.veri = veri;}
    }
```



Dizi tabanlı

Tam sayılar içeren bir kuyruğun sabit dizi ile tanımı

```
public class Kuyruk{
        public Eleman[] dizi;
        public int bas;
        public int son;
        int boyut;
        public Kuyruk(int boyut){
            dizi= new Eleman[boyut];
            this.boyut = boyut;
            bas = 0;
            son = 0;
        }
```

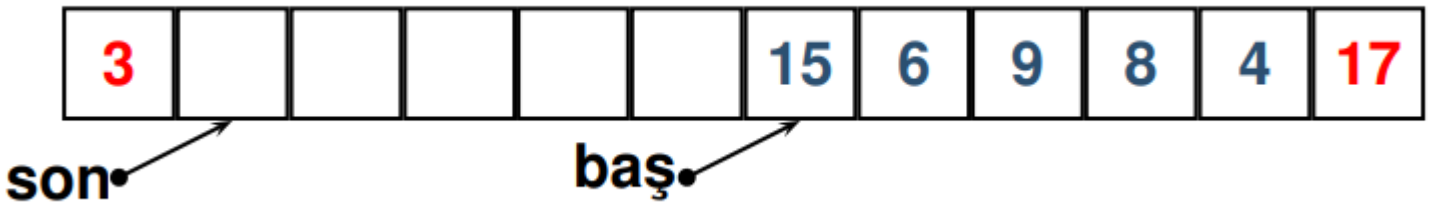
```
Boolean kuyrukDolu(){
            if (bas == (son + 1) % N)
                return true;
            else
                return false;}

        Boolean kuyrukBos(){
            if (bas == son)
                return true;
            else
                return false;}
}
```

Dizi tabanlı

Sabit dizi ile uygulanan bir kuyruga yeni bir eleman ekleyen algoritma

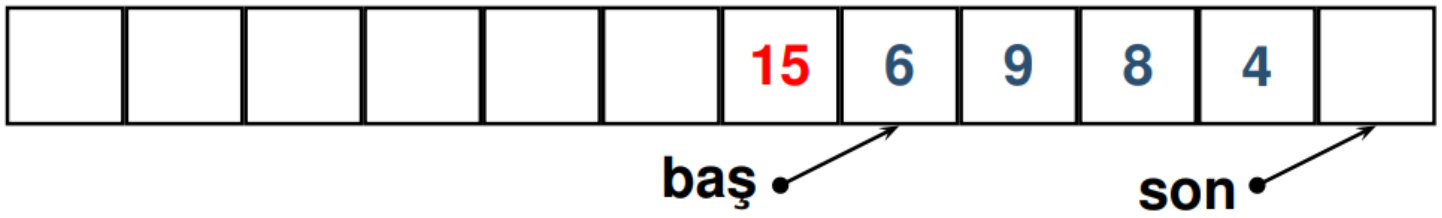
```
void kuyrugaEkle(Ornek yeni){
    if (!kuyrukDolu()){
        dizi [son] = yeni;
        son = (son + 1) % N;
    }
}
```



Dizi tabanlı

Sabit dizi ile uygulanan bir kuyruktan bir eleman silen algoritma

```
Ornek kuyrukSil(){
    Ornek sonuc;
    if (!kuyrukBos()){
        sonuc = dizi[bas];
        bas = (bas + 1) % N;
        return sonuc;
    }
    return null;
}
```



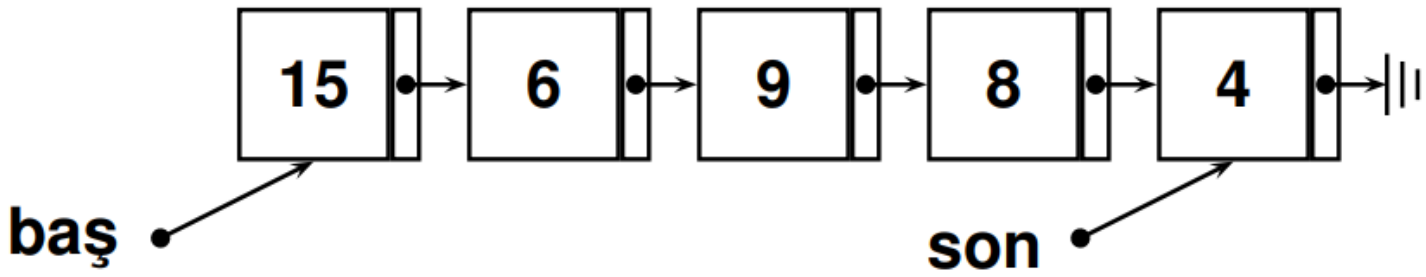
Dizi tabanlı

- Ekleme: O(1)
- Silme: O(1)
- Sınırlamalar
  - Kuyruğun eleman sayısı önceden tanımlanmalıdır ve değiştirilemez.
  - Dolu bir kuyruğa yeni bir nesne eklemeye çalışmak istisnai durumlara sebep olabilir.

Baglı Liste ile Queue Tanımı

- Tam sayılar içeren bir kuyruk için bağlı liste ile tanımı

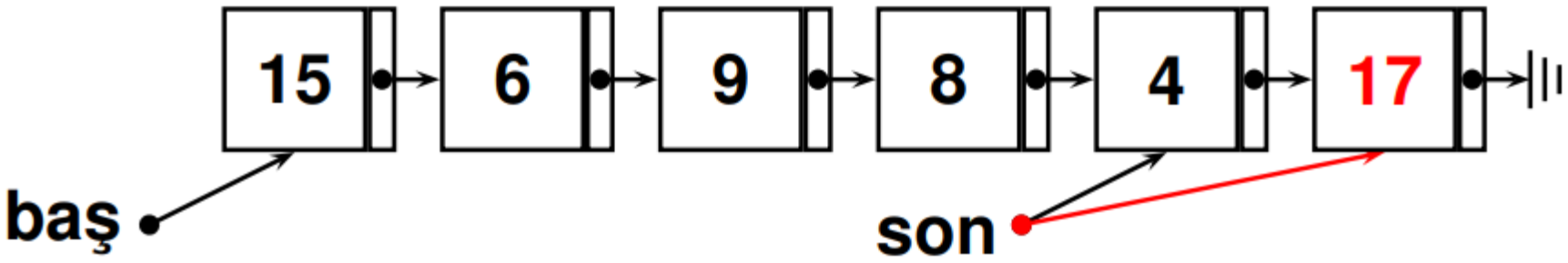
```
public class Kuyruk{
    Eleman bas;
    Eleman son;
    public Kuyruk(){
        bas = null;
        son = null;
    }
    boolean kuyrukBos(){
        if (bas == NULL)
            return true;
        else
            return false;
    }
}
```



Baglı Liste ile Queue Tanımı

- eleman ekleme

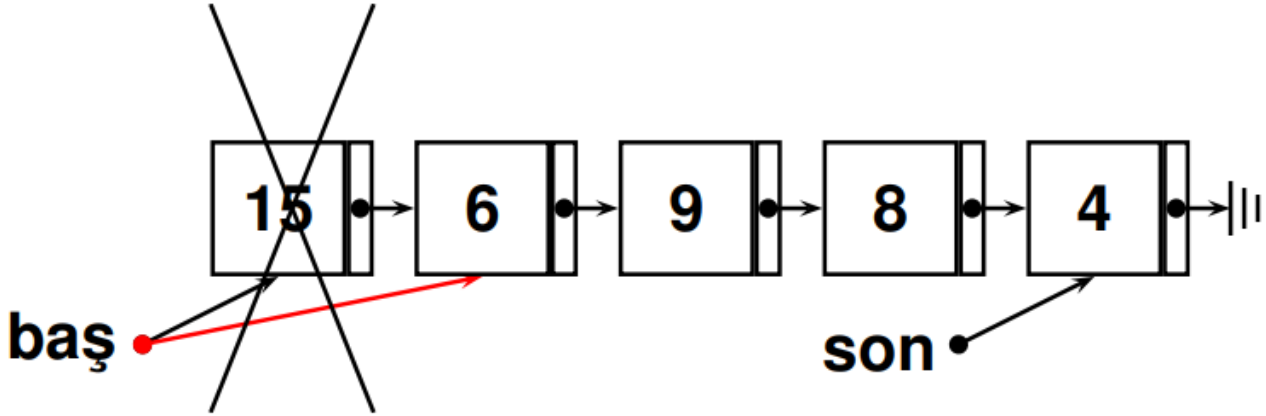
```
void kuyrugaEkle(Eleman yeni){
    if (!kuyrukBos())
        son. ileri  = yeni;
    else
        bas = yeni;
    son = yeni;
}
```



Baglı Liste ile Queue Tanımı

- eleman silme

```
Eleman kuyrukSil(){
    Eleman sonuc;
    sonuc = bas;
    if (!kuyrukBos()){
        bas = bas. ileri ;
        if (bas == null)
            son = null;
    }
    return sonuc;
}
```

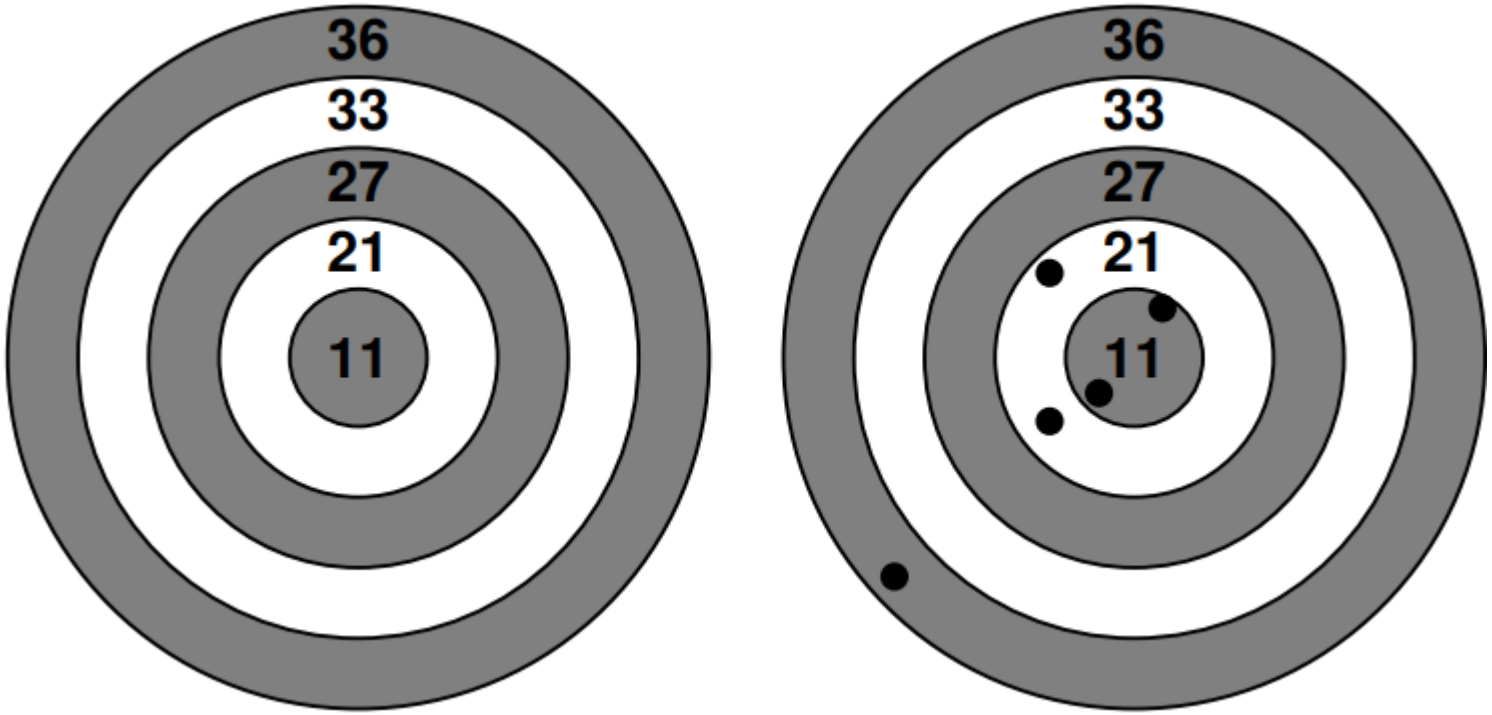


Ekleme: O(1)

Silme: O(1)

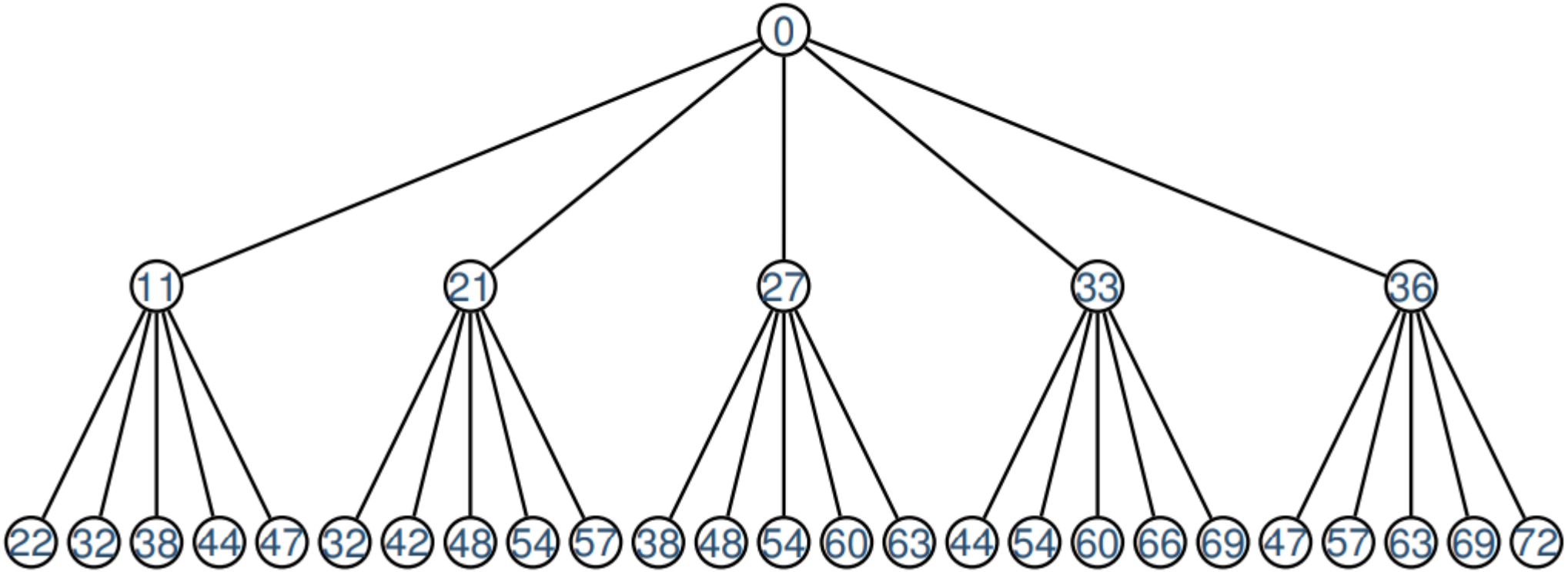
Baglı Liste ile Queue Tanımı

- Uygulama: Hedef Tahtası



Baglı Liste ile Queue Tanımı

- Hedef tahtası probleminde geniş arama yönteminin iki aşamasının uygulanması



Baglı Liste ile Queue Tanımı

- Hedef tahtası probleminde bir durumun tanımı

```
public class Eleman{
    int toplam;
    String  atis ;
    Eleman ileri ;
    public Eleman(int toplam, String atis){
        this.toplam = toplam;
        this.atis =  atis ;
         ileri   = null;
    }
}
```

Baglı Liste ile Queue Tanımı

- Hedef tahtası probleminin genis arama yöntemiyle çözümü

```
String hedefTahtasi(int[] tahta){
    int i, t;
    String a;
    Eleman e;
    Kuyruk k;
    e = new Eleman(0, "");
    k = new Kuyruk();
    k.kuyrugaEkle(e);
    while (!k.kuyrukBos()){
        e = k.kuyrukSil ();
        if (e.toplam == 100)
            return e.atis;
        for (i = 0; i < tahta.length; i++){
            if (e.toplam + tahta[i] <=100){
                t = e.toplam + tahta[i];
                a = e.atis + "␣" + tahta[i];
                e = new Eleman(t, a);
                k.kuyrugaEkle(e);
            }
        }
    }
    return null;
}
```

Ödev

- Palindrom (tersten okunuşu da aynı olan cümle, sözcük ve sayılar) (Ey Edip, Adana'da pideye, 784521125487 …vb).
- Verilen bir stringin palindrom olup olmadığını belirleyen C, C# kodunu, noktalama işaretleri, büyük harfler ve boşlukların ihmal edildiğini varsayarak stack ve queue yapılarıyla yazalım. (Hem bağlı liste ve hem de dizi ile tasarımı yapılmalıdır)
- Öncelikli kuyruk yapısının C, C# kodunu yazınız.
- Radix sort algoritmasını kuyruk kullanarak C, C# kodunu yazınız.