Bölüm 4. Kuyruk

Olcay Taner Yıldız

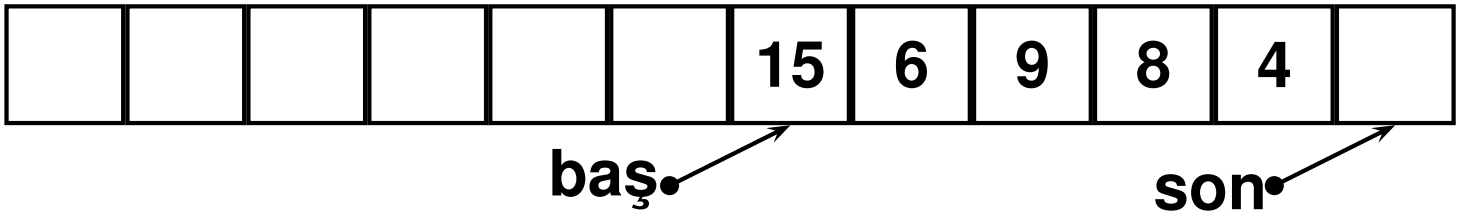
2014

Sabit Dizi ile Kuyruk Tanımı

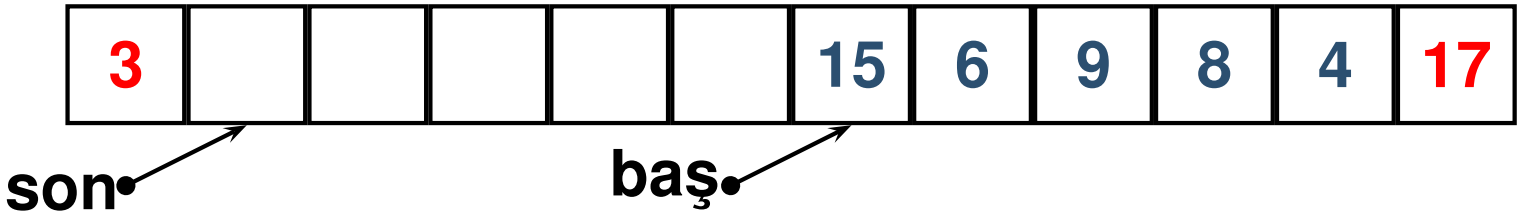
Tam sayılar içeren sabit dizi ile bir kuyruk uygulaması


```
1  public class Kuyruk{
2      Ornek dizi [];
3      int bas;
4      int son;
5      int N;
6      public Kuyruk(int N){
7          dizi  = new Ornek[N];
8          this.N = N;
9          bas = 0;
10         son = 0;
11     }
12     boolean kuyrukDolu(){
13         if (bas == (son + 1) % N)
14             return true;
15         else
16             return false;
17     }
18     boolean kuyrukBos(){
19         if (bas == son)
20             return true;
21         else
22             return false;
23     }
24 }
```

Azami 12 eleman alabilen bir kuyruk yapısı



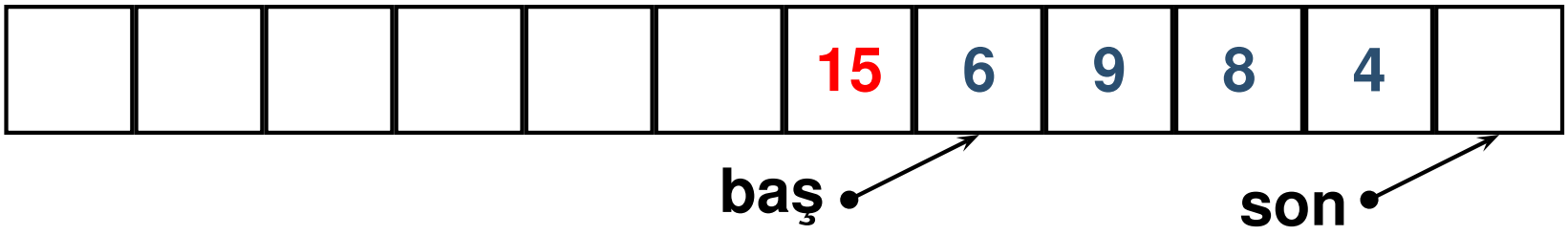
Önceki şekilde verilen kuyruk yapısına 17 ve 3 elemanlarının eklenmesi



Sabit dizi ile uygulanan bir kuyruğa yeni bir eleman ekleyen algoritma

```
1 void kuyrugaEkle(Ornek yeni){
2     if (!kuyrukDolu()){
3         dizi[son] = yeni;
4         son = (son + 1) % N;
5     }
6 }
```

Önceki şekilde verilen kuyruk yapısından 15'in silinmesi



Sabit dizi ile uygulanmış bir kuyruktan eleman silen ve o elemanı döndüren algoritma

```
1  Ornek kuyrukSil(){
2      Ornek sonuc;
3      if (!kuyrukBos()){
4          sonuc = dizi[bas];
5          bas = (bas + 1) % N;
6          return sonuc;
7      }
8      return null;
9  }
```

Kuyruk İşlemleri (Dizi)

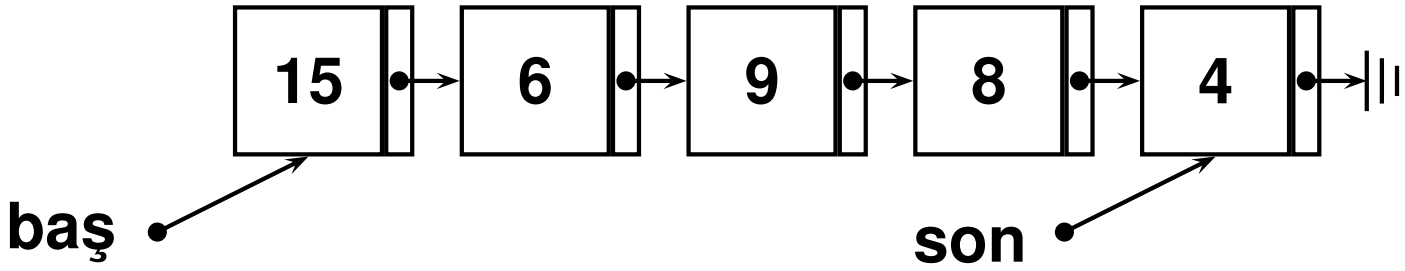
- Ekleme: $\mathcal{O}(1)$
- Silme: $\mathcal{O}(1)$

Bağlı Liste ile Kuyruk Tanımı

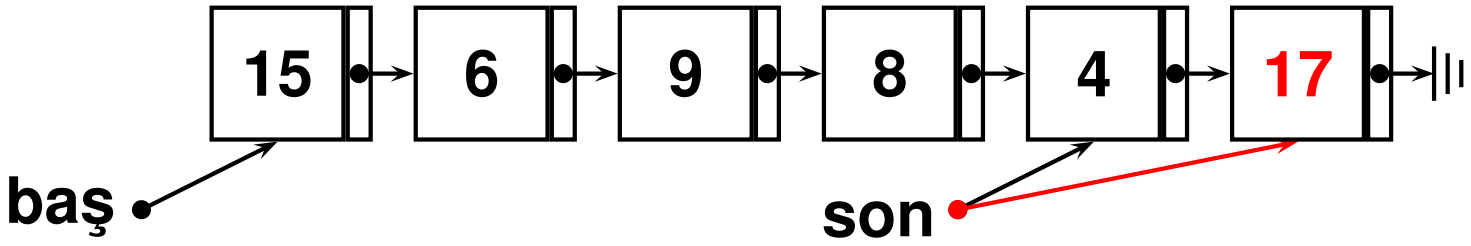
Tam sayılar içeren bağlı liste ile bir kuyruk uygulaması

```
1  public class Kuyruk{
2      Eleman bas;
3      Eleman son;
4      public Kuyruk(){
5          bas = null;
6          son = null;
7      }
8      boolean kuyrukBos(){
9          if (bas == NULL)
10             return true;
11         else
12             return false;
13     }
14 }
```

5 eleman içeren ve bağlı liste ile tanımlanmış kuyruk yapısı



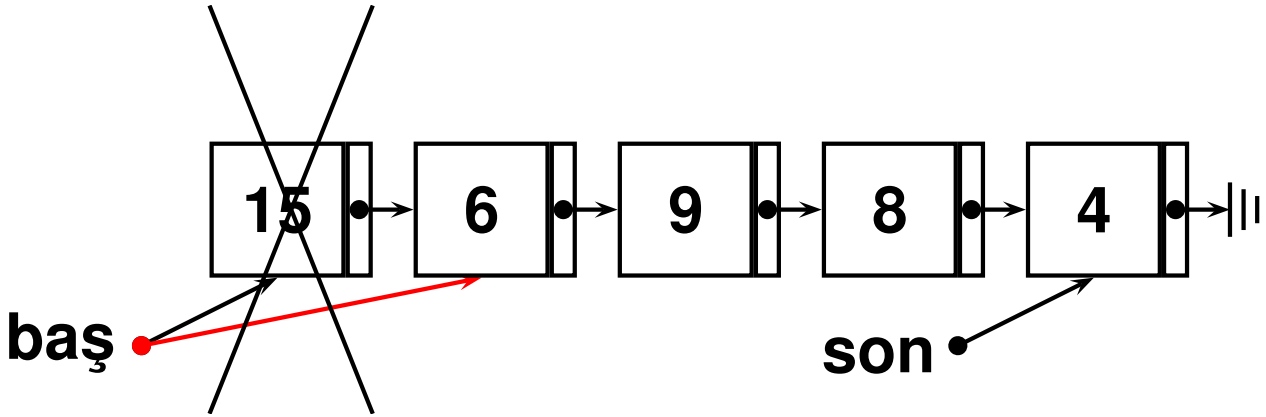
Önceki şekilde verilen kuyruk yapısına 17'nin eklenmesi



Bağlı liste ile uygulanan bir kuyruğa yeni bir eleman ekleyen algoritma

```
1  void kuyrugaEkle(Eleman yeni){
2      if (!kuyrukBos())
3          son. ileri  = yeni;
4      else
5          bas = yeni;
6      son = yeni;
7  }
```

Önceki şekilde verilen kuyruk yapısından 15'in silinmesi



Bağlı liste ile uygulanmış bir kuyruktan eleman silen ve o elemanı döndüren algoritma

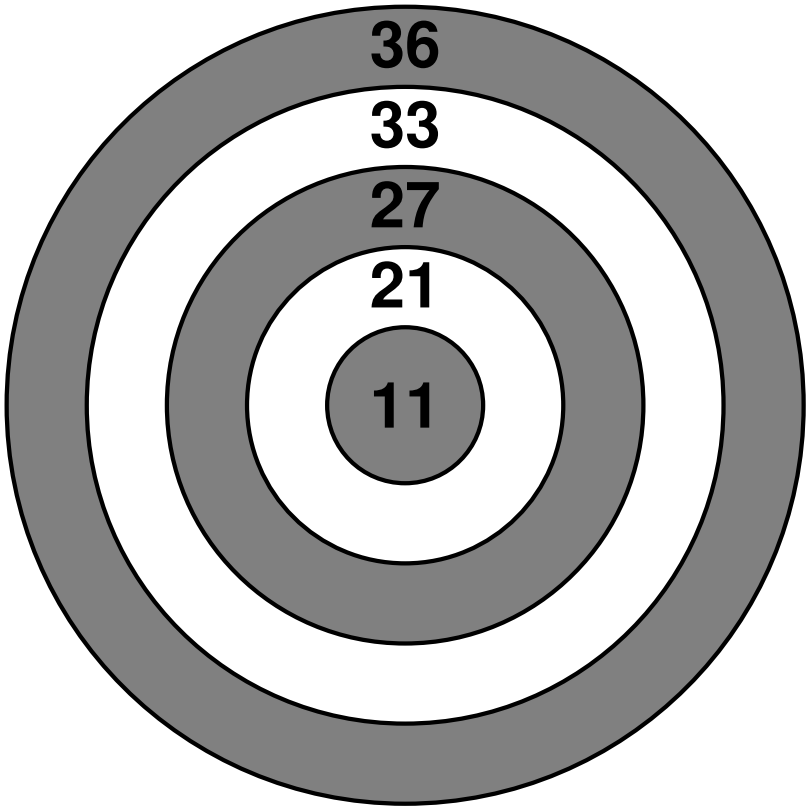
```
1  Eleman kuyrukSil(){
2      Eleman sonuc;
3      sonuc = bas;
4      if (!kuyrukBos()){
5          bas = bas. ileri ;
6          if (bas == null)
7              son = null;
8      }
9      return sonuc;
10 }
```

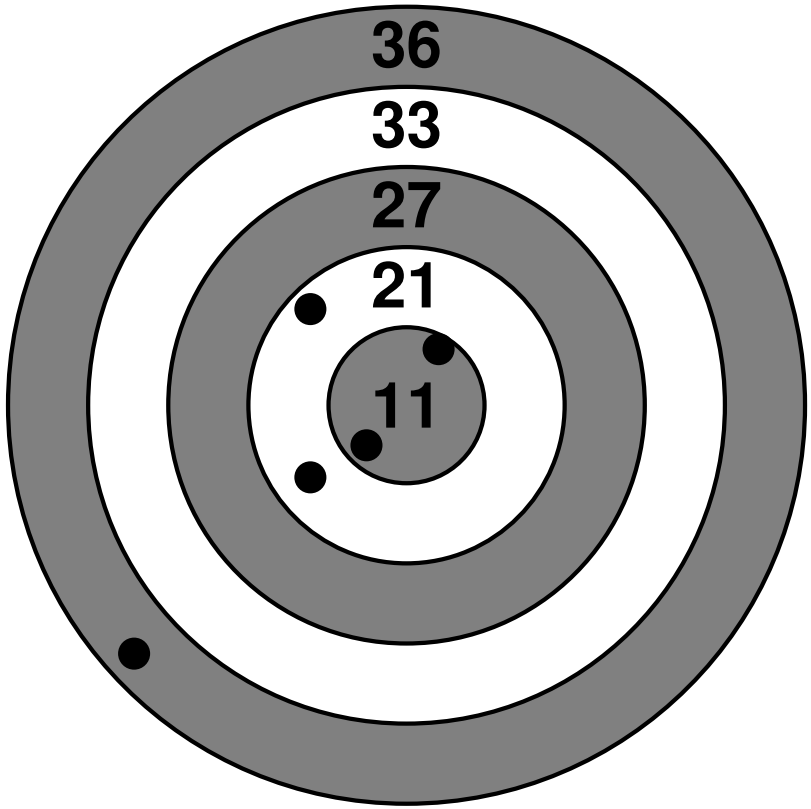
Kuyruk İşlemleri (Bağlı Liste)

- Ekleme: $\mathcal{O}(1)$
- Silme: $\mathcal{O}(1)$

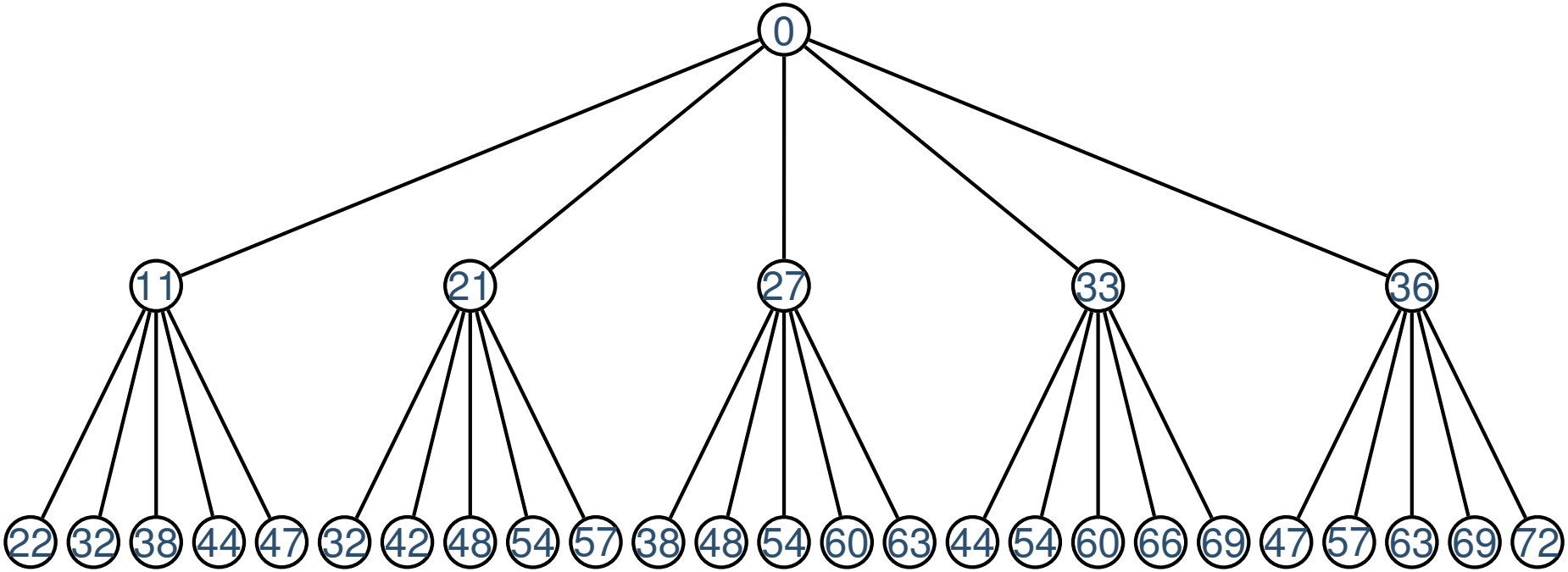
Uygulama: Hedef Tahtası

Örnek bir hedef tahtası ve yapılan 5 atışta 100 elde edilmesi





Hedef tahtası probleminde geniş arama yönteminin iki aşamasının uygulanması



Hedef tahtası probleminde bir durumun tanımı

```
1  public class Eleman{
2      int toplam;
3      String  atis ;
4      Eleman ileri ;
5      public Eleman(int toplam, String atis){
6          this.toplam = toplam;
7          this.atis =  atis ;
8           ileri   = null;
9      }
10 }
```

Hedef tahtası probleminin geniş arama yöntemiyle çözümü

```
1  String hedefTahtasi(int[] tahta){
2      int i, t;
3      String a;
4      Eleman e;
5      Kuyruk k;
6      e = new Eleman(0, "");
7      k = new Kuyruk();
8      k.kuyrugaEkle(e);
9      while (!k.kuyrukBos()){
10         e = k.kuyrukSil();
11         if (e.toplam == 100)
12             return e.atis;
13         for (i = 0; i < tahta.length; i++){
14             if (e.toplam + tahta[i] <=100){
15                 t = e.toplam + tahta[i];
16                 a = e.atis + "␣" + tahta[i];
17                 e = new Eleman(t, a);
18                 k.kuyrugaEkle(e);
19             }
20         }
21     }
22     return null;
23 }
```