Statik Yığıt ve Kuyruklar

2.1 Statik Yığıtlar

Burada statik yapıdan kasıt boyut değişi olmayan dizi mantığının kullanılmasıdır. Yığıtlar kullanımı en kolay liste yapılarındandır. Yığıtta ekleme ve çıkarma sadece bir uçtan yapılır ve bu yığıtın tepe kısmıdır.

Yığıt mantığı için genelde ekleme için itme (push) ve çıkarma için çekme (pop) deyimleri kullanılır. Programlama yapılırken kullanılan alt program çağırmalarında en çok kullanılan yöntemdir. Her alt program çağırıldığında CPU içerikleri yığıta itilir ve alt program bitiminde yığıttan çekilerek CPU'nun program koşumuna nerede kaldığı, değişkenlerin ne durumda olduğu hatırladır. En son giren ilk çıkar (LIFO: Last In Fırst Out) mantığını gerçekler.

Yığıt mantığının gerçekleştirilebilmesi için yığıtta saklanacak verileri tutacak bir diziye ve yığıtın en üst kısmını (son eklenen elemanı) işaret edecek bir değişkene ihtiyaç duyulur. Gerek yığıt ve gerekse başka zaman programlama yapılırken program, anlamlı olacak şekilde metodlar halinde verilmesi hem anlaşılırlığı artırır hem programın esnek ve yazılan metodların başka programlarda da kullanılması kolaylığını getirir. Bu mantık içerisinde yığıt mantığı kod olarak şöyle verilebilir.

```
// Stack.java
// demonstrates stacks
// to run this program: C>java StackApp
import java.io.*;                          // Giriş/Çıkış için
class StackX {
      private int maxSize;                 // Dizinin boyutu
      private double[] stackArray;
      private int top;                     // Yığıtın tepesi
      public StackX(int s){                // Yapılandırıcı
            maxSize = s;                   // En büyük dizi boyutunu belirle
            stackArray = new double[maxSize]; // Diziyi oluştur
            top = -1;                      // Yığıtta eleman yok
      }
      public void push(double data){       // Yığıta very itme
            if (isFull())
              System.out.println('Yığıt Dolu')
            else {
               top++; // top=top+1;
               stackArray[top]=data;       // Tepeyi artır, elemanı yerleştir.
            }
      }
      public double pop(){
            double temp=0.0;
            if (isEmpty()){
              System.out.println('Yığıt Boş')
            else {
               temp=stackArray[top];
               top--;
            }
            return temp;
```

```
    }
 public  boolean isFull(){
        return top==maxsize-1;
}
public boolean isEmpty(){
       return top==-1;
}
public int size(){
          return top+1;
}
public double peek(){
          return stackArray[top];
} // StackX sınıfı sonu
```

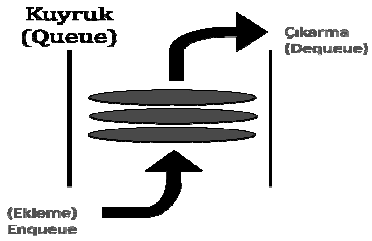
Yukarıdaki yığıt yapısını kullanan örnek bir program.

```
class StackApp
{
public static void main(String[] args)
{
        StackX theStack = new StackX(10);       // Yeni yığıt yap
        theStack.push(20);                      // Veriyi yığıta it
        theStack.push(40);
        theStack.push(60);
        theStack.push(80);

        while( !theStack.isEmpty() ){           // Boş olmadığı sürece yap
                double value = theStack.pop();  // Yığıttan eleman çek
        System.out.print(value);          // Bilgiyi ekrana yaz
        System.out.print("  ");
        }                                       // Döngü sonu
System.out.println(" ");
}                                               // main() sonu
}                                               // StackApp sınıfı sonu
```

2.2 Statik Kuyruklar

Kuyruk mantığına gerek günlük hayatta ve gerekse bilgisayarda sıraya sokulması olay veya işlerde oldukça sık karşılaşırız. Kuyruk yığıttan farklı olarak iki uçludur; bir başı ve bir sonu vardır. Bu nedenle bilgiyi saklamak için bir diziye ve en az iki indekse ihtiyaç vardır. Şekil 1'de bir kuyruk mantığı şematik olarak verilmiştir. Kuyruk mantığını gerçeklemede en uygun yaklaşım diziyi daireselmiş gibi düşüp eleman ekleme ve çıkarmada sürekli indis artırımına gitmektir.



Şekil 2.1. Kuyruk mantığını oluşması.

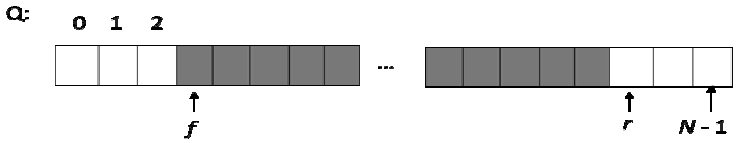
Kuyruk yapısını gerçeklemekteki temel fikir ilk girenin il çıkmasıdır (FIFO: First Int First Out). Kuyruk üzerinde yapılacak davranışlar aşağıdaki gibidir:

- Ekleme (Enqueue) : Kuyruğun sonuna eklenmesi
- Çıkarma (Dequeue) : Kuyruğun önünden bir çıkarılması.
- Ön(Front) : Eklenenlerin alındığı uç
- Arka(Rear) : Eklenen uç
- Dolu Olma (isFull) : Dolu mu?
- Boş Olma (isEmpty) : Boş mu?
- Başlatış (Initialize) : Boş kuyruk hazırlama
-

Kuyruk üzerindeki işlemler örnek aşağıdaki gibi verilebilir.

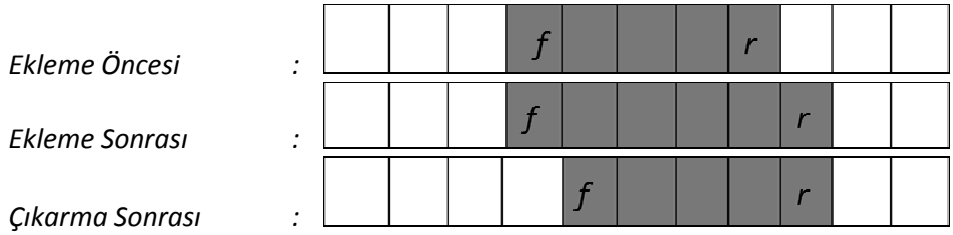
İşlem	Çıkış	front ←Q← rear
enqueue(5)	-	(5)
enqueue(3)	-	(5,3)
dequeue()	5	(3)
enqueue(7)	-	(3,7)
dequeue()	3	(7)
front()	7	(7)
dequeue()	7	()
dequeue()	“error”	()
isEmpty()	true	()
enqueue(9)	-	(9)
enqueue(7)	-	(9,7)
size()	2	(9,7)
enqueue(3)	-	(9,7,3)
enqueue(5)	-	(9,7,3,5)
dequeue()	9	(7,3,5)

Kuyruk yapısı şematik olarak şöyledir. Veriyi saklamak için Q(ueue) gibi bir dizi ile ilk işlem görecek elemanı gösteren bir f(ront) indisi ve son eklenen elemanı gösteren bir r(ear) indisi mevcuttur.



Şekil 2.2. Bir kuyruk yapısı.

Eğer dizi veri eklemeden önce şekil 2.3a’daki gibi ise ekleme sonucu şekil 2.3b’deki gibi ve çıkarma sonucu ise şekil 2.3c’deki gibi olur.

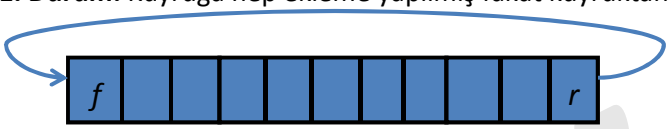


Şekil 2.3.Kuyruğa ekleme ve kuyruktan çıkarma.

Burada dizi dairesel gibi düşünüldüğünde iki indis ile kuyruğun boş veya dolu olduğu anlaşılamayacaktır. Şöyleki kuyruk için uç durumlar olan tam dolu veya tam boş olması analizi yapılırsa durum çok net olarak görülecektir.

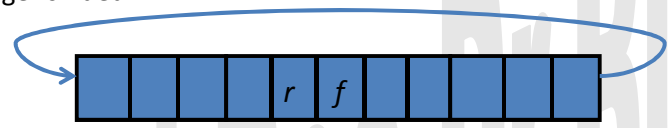
Tam dolu olma durumları:

1. Durum: Kuyruğa hep ekleme yapılmış fakat kuyruktan hiç çıkarma olmamıştır.



f=0 ve r=Max-1

2. Durum: Kuyruğa ekleme ve kuyruktan çıkarmalar olmuş ve r(ear) indeksi dairesellikten dolay dizi sonundan dizi başına geçerek bir tur atmıştır. Sonra yine eklemeler olmuş ve f(ront)un bir adım gerisindedir.



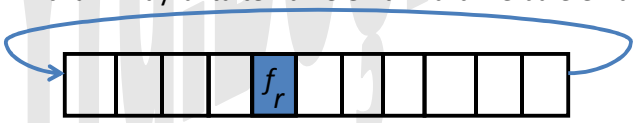
front=rear+1

Tam boş olma durumları:

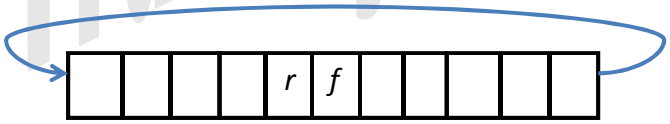
1. Durum: Kuyruğa hiç eleman eklemesi yapılmamıştır. İnteksler başlangış durumundadır. olmamıştır.

f=0 ve r=-1

2. Durum: Kuyrukta tek bir eleman vardı ve bu elemanda çıkarıldıktan sonraki durumdur.



Tek bir eleman mevcut



front=rear+1

Dikkat edilirse hem tamamen boş olma hem de tamamen dolu olma durumlarında ***front=rear+1*** şartı gerçekleşmektedir. Bu durumda sadece front ve rear (ön ve arka) gibi iki indeksle kuyruğun boş veya doluluğunu doğrudan test etmek mümkün değildir.
İşte bu durumda en pratik çözüm kuyruk içindeki veriyi saymaktır. Bu da üçüncü bir sayıcı değişkeni tanımlamak ile mümkündür. Aşağıda sayıcı kontrollü kuyruk yapısı için java sınıfı verilmiştir:

```
public arrayQueue {
        public static final int CAPACITY = 1000;
        private int maxSize;                    //En büyük dizi boyutu
        private Object [] arrayQ;               // Verileri tutacak dizi
        private int front=0;                    //İlk işlem görecek eleman indeksi
        private int rear=-1;                    //Son işlem görecek eleman indeksi
        private int counter=0;                  // Sayıcı
        public arrayQueue() {                   //Parametresiz yapılandırıcı
                this( CAPACITY );
        }
        public ArrayQueue( int Max ) {          //Parametreli yapılandırıcı
                maxSize = Max;
                arrayQ = new Object[maxSize ];
        }
```

```
        public void enqueue( Object element ){
                if (counter==maxSize)
                   System.out.println("Kuyruk Dolu")
                else{
                   rear=(rear+1)%maxSize;
                   arrayQ[rear]=element;
        }
        public Object dequeue()  {
                    Object element=null;
                   if (counter==0)
                            System.out.println("Kuyruk Boş")
                   else{
                            element=Q[front];
                            arrayQ[front]=null;
                            front=(front+1)%maxSize;
                   }
           return element;
        }
    public int size() {
        return counter;
    }
    public boolean isEmpty(){
        return capacity==0;
    }
    public Object front(){
       if (counter==0)
         return null
     else
         return arrayQ[front];
    public Object rear(){
       if (counter==0)
         return null
     else
         return arrayQ[rear];

}// Kuyruk sınıfının sonu
```

Yukarıda sayıcı kontrollü kuyruk yapısı verilmiştir. Bunun yerine kuyruğun boşalması durumu özel kabul edilerek indeksler bu özel değerlere ayarlanabilir. Özel indeks değerli kuyruk diyebileceğimiz bu mantığın kodlaması ise aşağıdaki gibi yapılabilir.

```
public class ArrayQueue implements Queue {
        public static final int CAPACITY = 1000;
        private int maxSize;
        private Object [] arrayQ;
        private int front=0;
        private int rear=-1;
        public ArrayQueue() {
                this( CAPACITY );
        }
        public ArrayQueue( int Max ) {
                maxSize = Max;
```


Yrd.Doç.Dr.Burhan ERGEN

```
            arrayQ = new Object[maxSize ];
    }
    public void enqueue(Object element){

      if ((front==0&&rear==maxSize-1)||(rear>0&&rear+1==front))
         System.out.println("Kuyruk Dolu")
      else{
            rear=(rear+1)%capacity;
            arrayQ[rear]=element;
      }
public Object dequeue() {
      Object element=null;
      if (rear==-1)
          System.out.println("Kuyruk Boş")
       else { element=arrayQ[front];
            if (front==rear) {
               front=0;
               rear=-1;
             }
            else
                front=(front+1)%maxSize;
       return element;
    }

public int size() {
    if (rear>front)
        return  (front-rear+1)
    else
         return (maxSize-(front-rear)+1);
    }
public boolean isEmpty(){
    return rear==-1;
}
public boolean isFull(){
return  (front==0&&rear==maxSize-1)||(rear>0&&rear+1==front);
    }

public Object front(){
   if (rear==-1)
    return null
else
    return arrayQ[front];
}
public Object rear(){
   if (rear==-1)
    return null
else
    return arrayQ[front];
}
```

Yrd.Doç.Dr.Burhan ERGEN