KUYRUKLAR
QUEUES

Doç. Dr. Aybars UĞUR

Giriş

Bu bölümde gerçek yaşamdaki kuyrukların bilgisayardaki gösterimleri üzerinde durulacaktır. Kuyruklar, eleman eklemelerin sondan (rear) ve eleman çıkarmaların baştan (front) yapıldığı veri yapılarıdır. Bir eleman ekleneceği zaman kuyruğun sonuna eklenir. Bir eleman çıkarılacağı zaman kuyrukta bulunan ilk eleman çıkarılır. Bu eleman da kuyruktaki elemanlar içinde ilk eklenen elemandır. Bu nedenle kuyruklara **FIFO (First-In First-Out = ilk giren ilk çıkar)** listeleri de denilmektedir.

Gerçek yaşamda da bankalarda, duraklarda, gişelerde, süpermarketlerde, otoyollarda kuyruklar oluşmaktadır. Kuyruğa ilk olarak girenler işlemlerini ilk olarak tamamlayıp kuyruktan çıkarlar. Veri yapılarındaki kuyruklar bu tür veri yapılarının simülasyonunda kullanılmaktadır. Ayrıca işlemci, yazıcı, disk gibi kaynaklar üzerindeki işlemlerin yürütülmesinde ve bilgisayar ağlarında paketlerin yönlendirilmesinde de kuyruklardan yararlanılmaktadır.

Kuyruk İşlemleri ve Tanımları

insert(q,x) **:** q kuyruğunun sonuna x elemanını ekler. (enqueue)

x=remove(q) **:** q kuyruğunun başındaki elemanı silerek x'e atar. (dequeue)

Bir Kuyruk Sınıfı Tasarımı
C# ile Gerçekleştirimi - I

```
class Kuyruk
{ // Diziler üzerinde bir double kuyruğu
  private int boyut;
  private double[] kuyrukDizi;
  private int baş, son;
  private int elemanSayısı;

  public Kuyruk(int s) // Yapılandırıcı
  { boyut = s;
    kuyrukDizi = new double[boyut];
    baş = 0; son = -1; elemanSayısı = 0;
  }

  public void enque(double j) // Kuyruk sonuna eleman ekler
  { if(son == boyut-1) // başa dönme durumu
      son = -1;
    kuyrukDizi[++son] = j; // sonu arttır ve ekle
    elemanSayısı++;
  }
```

Bir Kuyruk Sınıfı Tasarımı
C# ile Gerçekleştirimi - II

```
  public double deque() // Kuyruğun başından bir eleman çıkarır
  {
    double temp = kuyrukDizi[baş++]; // Değeri alıp başı arttır
    if(baş == boyut) // başa dönme durumu
      baş = 0;
    elemanSayısı--;
    return temp;
  }

  public bool bosMu() // true, Kuyruk boş ise
  {
    return (elemanSayısı==0);
  }

} // end class Kuyruk
```

Tasarlanan Kuyruk Sınıfının Kullanımı

1'den 10'a kadar olan sayıları kuyruğa yerleştirip sırayla çıkaran program

```
class Program
{
  static void Main(string[] args)
  {
    Kuyruk k = new Kuyruk(25);
    k.enque(1); k.enque(2);
    Console.WriteLine(k.deque()); // 1
    k.enque(3);
    for(int i=4; i<10; ++i)
      k.enque(i);
    while(!k.bosMu())
      Console.WriteLine(k.deque());
  }
}
```

C# Hazır Kuyruk Sınıfının Kullanımı

```
static void Main(string[] args)
{
  Queue kuyruk = new Queue();
  kuyruk.Enqueue("Bir"); kuyruk.Enqueue("İki"); kuyruk.Enqueue("Üç");
  yazdir(kuyruk);
  string str = (string) kuyruk.Dequeue();
  yazdir(kuyruk);
}
static void yazdir(IEnumerable koleksiyonum)
{
  foreach (Object obj in koleksiyonum)
    Console.Write("{0} ", obj);
  Console.WriteLine();
}
```

Ekran Çıktısı :
Bir İki Üç
İki Üç

C# ortamında
Generic Queue sınıfı da vardır.

Java ile Bir Kuyruk Sınıfı

```
// Diziler üzerinde bir tamsayı kuyruğu

class Kuyruk
{
  private int boyut;
  private int[] kuyrukDizi;
  private int bas;
  private int son;
  private int elemanSayisi;

  // Yapılandırıcı Metot (Constructor)
  public Kuyruk(int s)
  {
    boyut = s;
    kuyrukDizi = new int[boyut];
    bas = 0;
    son = -1;
    elemanSayisi = 0;
  }

public void ekle(int j) // Kuyrugun sonuna
      eleman ekler
  {
    if (son==boyut-1) son = -1;
    kuyrukDizi[++son] = j;
    elemanSayisi++;
  }

  public int cikar()
  {
    int temp = kuyrukDizi[bas++];
    if(bas==boyut) bas=0;
    elemanSayisi--;
    return temp;
  }

  public boolean bosMu()
  {
    return(elemanSayisi==0);
  }
}
```

Hazırlanan Kuyruk Sınıfının Kullanımı

```
// 1'den 10'a kadar olan sayilari kuyruga yerlestirip
// sırayla çıkaran program

public class KuyrukTest
{
  public static void main(String args[])
  {
    Kuyruk k = new Kuyruk(25);

    k.ekle(1);
    k.ekle(2);
    System.out.println(k.cikar()); // 1
    k.ekle(3);
    for(int i=4; i<10; ++i)
      k.ekle(i);
    while(!k.bosMu())
      System.out.println(k.cikar());
  }

}
```

Java Hazır Kuyruk Sınıfının Kullanımı

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {

    public static void main(String[] args) {

        Queue<String> q=new LinkedList<String>();

        q.add("A1");
        q.add("A2");
        q.add("A3");
        q.add("A4");
        q.add("A5");

        Iterator itr=q.iterator();

        System.out.println("Initial Size of Queue :"+q.size());

        while(itr.hasNext())
        {
            String iteratorValue=(String)itr.next();
            System.out.println("Queue Next Value
           :"+iteratorValue);
        }

        // Kuyruktaki ilk değeri alır
        System.out.println("Queue peek :"+q.peek());
        // Kuyruktaki ilk değeri alır ve kuyruktan çıkarır
        System.out.println("Queue poll :"+q.poll());

        System.out.println("Final Size of Queue
           :"+q.size());
    }
}
```

```
Initial Size of Queue :5
Queue Next Value :A1
Queue Next Value :A2
Queue Next Value :A3
Queue Next Value :A4
Queue Next Value :A5
Queue peek :A1
Queue poll :A1
Final Size of Queue :4
```

Öncelik Kuyruğu - I

- Yığıtlarda ve kuyruklarda elemanlar eklenme sırasına dayalı olarak sıralanırlar. Yığıtlarda ilk olarak son eklenen, kuyruklarda ise ilk eklenen eleman çıkarılır. Elemanlar arasındaki gerçek sıralama (sayısal sıra veya alfabetik sıra gibi) dikkate alınmaz.

- Öncelik kuyrukları, temel kuyruk işlemlerinin sonuçlarını elemanların gerçek sırasının belirlediği veri yapılarıdır. Azalan ve artan sırada olmak üzere iki tür öncelik kuyruğu vardır. Artan öncelik kuyruklarında elemanlar herhangi bir yere eklenebilir ama sadece en küçük eleman çıkarılabilir. apq, artan öncelik kuyruğu olmak üzere pqinsert(apq,x) x elemanını kuyruğa ekler ve pqmindelete(apq) en küçük elemanı kuyruktan çıkararak değerini döndürür. Azalan öncelik kuyruğu ise artan öncelik kuyruğunun tam tersidir.

Öncelik Kuyruğu - II

Artan öncelik kuyruğunda önce en küçük eleman, sonra ikinci küçük eleman sırayla çıkarılacağından dolayı elemanlar kuyruktan artan sırayla çıkmaktadırlar. Birkaç eleman çıkarıldıktan sonra ise daha küçük bir eleman eklenirse doğal olarak kuyruktan çıkarıldığında önceki elemanlardan daha küçük bir eleman çıkmış olacaktır.

Öncelik kuyruklarında sadece sayılar veya karakterler değil karmaşık yapılar da olabilir. Örnek olarak telefon rehberi listesi, soyad, ad, adres ve telefon numarası gibi elemanlardan oluşmaktadır ve soyada göre sıralıdır.

Öncelik kuyruklarındaki elemanların sırasının elemanların alanlarından birisine göre olması gerekmez. Elemanın kuyruğa eklenme zamanı gibi elemanların alanları ile ilgili olmayan dışsal bir değere göre de sıralı olabilirler.

Öncelik kuyruklarının gerçekleştiriminde dizi kullanımı etkin bir yöntem değildir.

ALIŞTIRMALAR

- C# ile Aşağıdaki **Öncelik Kuyruğu** Sınıflarını Tasarlayıp Yazınız (Altyapıda **dizi**, **ArrayList** ve **List** yapılarını kullanabilirsiniz):
  - a) Tüm elemanları string olan
  - b) Tüm elemanları double olan
  - c) Tüm elemanları Ogrenci (no, ad, soyad, ort) sınıfı şeklinde olan
    - Ada göre alfabetik sırada öncelikli
    - Ortalamaya göre yüksekten düşüğe göre öncelikli

Java'daki Hazır Öncelik Kuyruğunun Kullanımı

```
import java.util.*;

public class PriorityQueueDemo {

    public static void main(String[] args){

        PriorityQueue<String> stringQueue = new PriorityQueue<String>();

        stringQueue.add("Masa");
        stringQueue.add("Sıra");
        stringQueue.add("Sandalye");
        stringQueue.add("Dolap");

        Iterator<String> itr = stringQueue.iterator();
            while(itr.hasNext())
                System.out.println(itr.next()); // Dolaşma
        while(stringQueue.size() > 0)
            System.out.println(stringQueue.remove()); // Silme
    }
}
```

```
Dolap
Masa
Sandalye
S²ra
Dolap
Masa
Sandalye
S²ra
```

Sınıfın Bir Üyesine Göre (ad) Öncelik Kuyruğu

```
import java.util.Comparator;
import java.util.PriorityQueue;

class Urun {
        String ad;
        int fiyat;
        public Urun (String adi, int fiyati)
        { ad = adi; fiyat = fiyati;                }
}


class UrunComparator implements Comparator
{
    public int compare(Object v1,Object v2)
    {
        return ((Urun)v1).ad.compareTo(((Urun)v2).ad);
    }
}
```

```
public class Test
{
    public static void main(String[] args)
    {
        Comparator comparator = new UrunComparator();
        PriorityQueue kuyruk =
            new PriorityQueue(10,comparator);
        kuyruk.add(new Urun("Masa", 10));
        kuyruk.add(new Urun("Dolap", 15));
        kuyruk.add(new Urun("Sandalye", 5));
        while (kuyruk.size() != 0)
        {
            Urun u = (Urun)kuyruk.remove();
            System.out.println(u.ad+" "+u.fiyat);
        }
    }
}
```

```
Dolap 15
Masa 10
Sandalye 5
```

Alıştırmalar

- Urun sınıfının ad üyesine göre büyükten küçüğe sıralı bir öncelik kuyruğu oluşturacak şekilde UrunComparator sınıfını değiştiriniz.

- Urun sınıfının fiyat üyesine göre küçükten büyüğe sıralı bir öncelik kuyruğu oluşturacak şekilde UrunComparator sınıfını değiştiriniz.