Çift Yönlü Bağlı Listeler ve Kuyruklar

4.1 Çift Yönlü Bağlı Liste

Bağlı listelerde en önemli olan nokta bağ yapısının nasıl oluşturulduğudur. Daha önce anlatılan bağlı listelerde bağlantıyı sağlayan tek bir referans vardı. Bu bağ sayesinde liste başından sonuna kadar ilerlemek mümkündü. Liste sonu bu şekilde organize edilmiş listelerde, kuyruk yapısında olduğu gibi, bilinebilmekle beraber liste sonundan liste başına ilerlemek, bir önceki düğüme erişmek mümkün değildir.

Bağlantılı listelerdeki bu durum dikkate alınırsa akla gelen ilk soru, bir düğümde birden fazla bağ oluşturulabilip oluşturulamayacağıdır. Elbette ki birden fazla, istenildiği kadar bağ oluşturulabilir. Daha da ötesi bağ yapısı sağlayan bütün referansların aynı türden olma zorunluluğu da yoktur. Yani bir düğüm yapısı içerisinde farklı tür düğüm yapılarını işaret edebilecek, karmaşık bir yapı oluşturmak mümkündür.

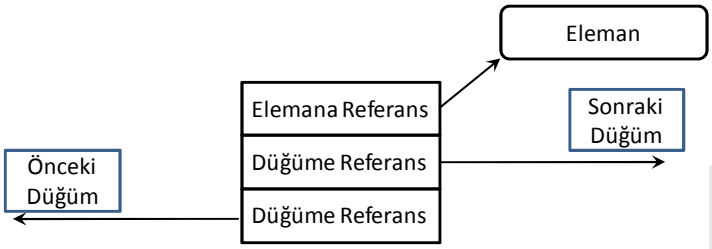
Java programlama dilinde Object sınıfı ile diğer standart dillerden bir farklı olarak her türlü yapıyı sergileyebilir. Bu bakımdan dilden bağımsız düşünülürse, bir düğüm içerisinde çok farklı düğümleri gösterecek alanlar bulunabilir. Bu kullanım açısından oldukça esneklik tanırken, her hangi bir düğüm çıkarıp eklenmesi durumunda, uygunca koparılıp eklenmesi gereken bir çok bağ olmasına neden olur. Yani bir düğüm için dikkatli davranılması ve kontrol edilmesi gereken bir çok bağlantı oluşmasıdır ki bu da kod yazımını zorlaştırır.

Bu nedenle bağ sayısının artması çok da istenen bir durum değildir. Fakat yine de bağ üzerinde ileri/geri hareket edebilmek birçok açıdan esneklik sağlayabilecek bir aranan bir yapıdır.

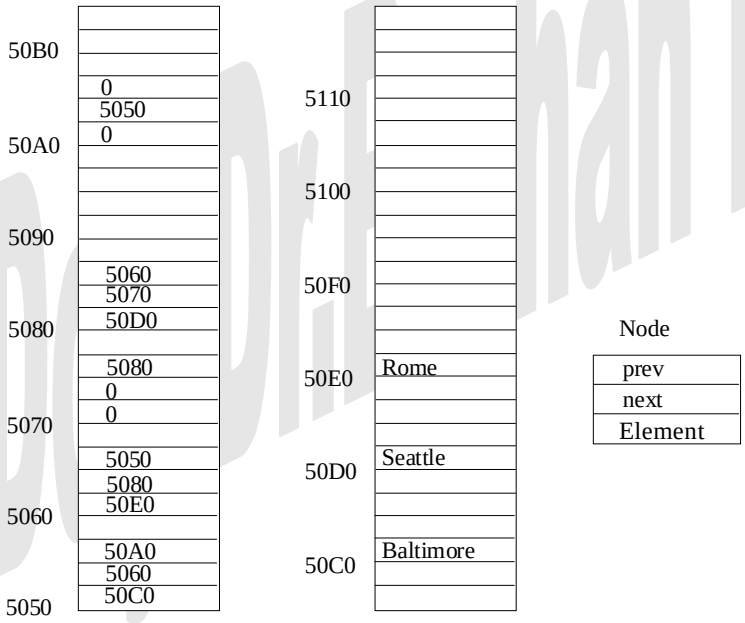
İleri ve geri ilerleme yönü sağlayabilecek düğüm sınıfı java için şöyle verilebilir.

```
public class dnode{
        object element;
        dnode next;         //Sonraki düğümü gösterecek referans
        dnode prev;         //Önceki düğümü gösterecek referans (previous)
        public dnode(){
                this(null,null,null);
        }
        public dnode(Object obj, dnode p, dnode n){
                element=obj;
                prev=p;
                next=n;
        }
}
```

Şekil 4.1 de çift yönlü düğüm yapısının mantığı şema olarak verilmiştir.



Şekil 4.1. Çift yönlü düğüm yapısı.



Şekil 4.2. Çift yönlü düğüm yapısının bellekteki durumu.

Daha önce kuyruk yapısında görüldüğü gibi eğer listenin sonuna ekleme yapılmak istenirse ilk düğüm eklemesi ile sonraki eklemeler arasında mantık ve kodlama açısından farklılık vardır. İlk düğümün eklemesinde daha önce düğüm olmadığı için ekleme yapılırken sonu gösterecek referansa da bağlantı yapma gerekliliği vardır. Çünkü tek bir düğüm var olacağından öncesi ve sonrası yoktur. Dolayısıyla her iki referansta bu düğümü göstermelidir.

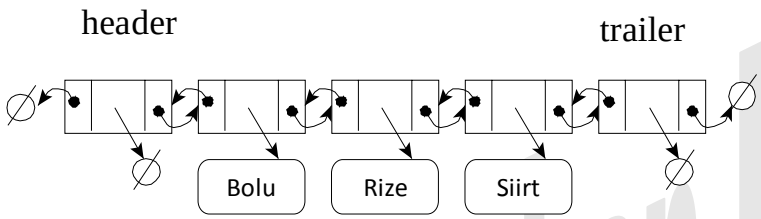
Fakat sonraki eklemelerde sona ekleme yapılacağından etkilenen sadece sonu gösteren (tail veya rear) indeksi olacaktır.

Çift yönlü bağlı listeler dikkate alındığında her bir düğümün hem öncesi (previous) hem de sonrası (next) vardır. Yani yukarıdaki düğüm yapısına göre; her bir düğümün öncesine ve sonrasına eklemeler olabilir.

Bu durumda; liste başına ekleme, liste sonuna ekleme, düğümün öncesine ekleme, düğümün sonrasına ekleme olmak üzere dört durum söz konusu olur. Bu şekilde düğüm ekleme ve çıkarma mantıklarını kurmak ve kodlamak oldukça karmaşık olacaktır. Karmaşıklığa neden olan durum öncesinin veya sonrasının olmamasıdır. Eğer her düğümün öncesi ve sonrası olursa eklemeler genel olarak aynı mantıkta ve çıkarmalar yine genel olarak aynı mantıkta olacaktır. Bağlı listede her bir düğümün öncesi ve sonrası olursa, ekleme yapılırken aslında yapılan iki düğüm arasına bir düğüm yerleştirmekten ibaret olacaktır. Çıkarmada ise yapılacak işlem her bir düğüm alınması iki düğüm arasından bir düğümün alınması işlemi olacaktır.

Bahsedilen bu kolaylıktan faydalanmak için çift yönlü bağlı liste oluşturmada; liste başında ve liste sonunda hiçbir eleman tutmayan bekçi görevli düğümler oluşturulabilir. Yani listenin ilk ve son

düğümü boş düğümlerdir. Şekil 4.3'te başında ve sonunda boş bekçi düğümleri olan çift yönlü bağlı liste yapısı verilmiştir.

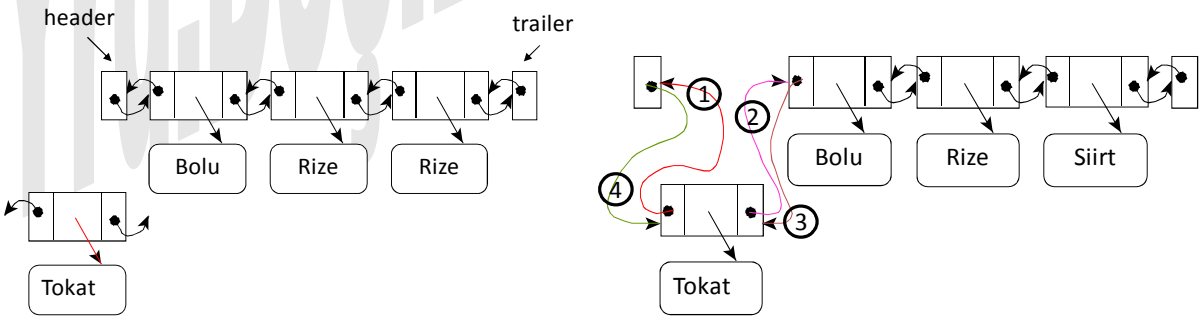


Şekil 4.3. Başında ve sonunda boş düğüm olan çift yönlü bağlı liste.

- **Başa Ekleme:**

Ekleme mantığı her hangi bir yer için aynı olmakla beraber kullanılan referanslar ve bağlantı esnasında kullanılacak ileri ve geri düğümler farklılaşır. Şekil 4.4'te eklemeden önceki ve sonraki bir bağlı liste yapısı verilmiştir. Oluşturulan yen düğüm liste başına yani header ile header.next arasına yerleştirilecektir. İlk düğüm boş kabul edildiğinden eklenen düğüm ilk düğüm olacaktır. Yeni düğüm şöyle oluşturulabilir.

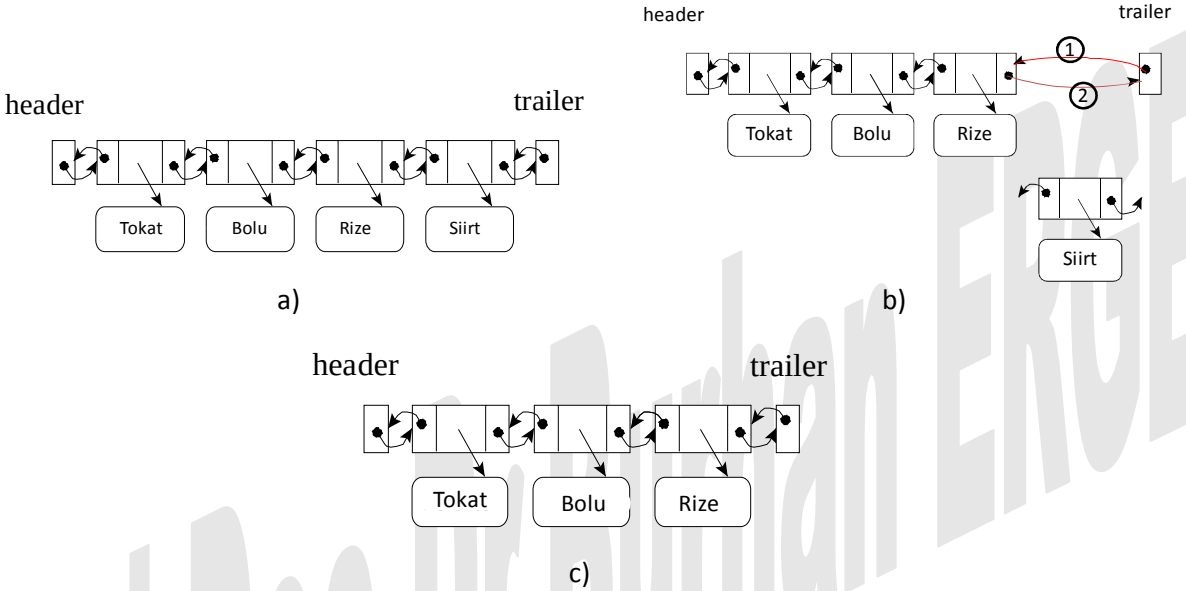
```
dnode newNode = new dnode();
newNode.element(new String("Tokat"));
```



Şekil 4.4. Yeni düğümün başa eklenmesi; a) Eklemeden önce, b) Eklemeden sonra.

Şekil 4.4b deki bağlar numarası sırasına göre kod olarak şöyle verilebilir:

1. newNode.prev= header;
2. newNode.next= header.next;
3. header.next.prev=newNode;
4. header.next=newNode;



Şekil 4.5. Sondan bir düğüm çıkarılması; a) Çıkarmadan önce, b) Çıkarma, c) Çıkarmadan sonra.

Çıkarma işlemleri şekil 4.5b deki bağ durumu numaralarına göre verilirse şöyle verilebilir:

1. trailer.prev.prev.next=trailer;
2. trailer.prev=trailer.prev.prev;

Çift yönlü bir bağlı liste genelde kullanılabilecek yapısı ve kullanılacak metodların kodları aşağıda verilmiştir. Düğüm yapısı olarak yukarıda tanımlanan çift yönlü düğüm yapısının kullanıldığı kabul edilmiştir.

```
public class DlinkedList {
    int numElems;                          // Listedeki eleman sayısı
    dnode header, trailer;                 // Baş ve son düğümleri /bekçi düğümler
    public DlinkedList() {
          numElts = 0;
          header = new dnode(null, null, null);      // Header oluştur.
          trailer = new dnode(null, header, null);   // Trailer oluştur.
          header.next=trailer;                       //Header ve Trailer birbirlerini gösteriyor.
    }

public int size(){
         return numElts;
         }
public boolean isEmpty() {
        return (numElts == 0);
 }
public Object first()
   if (isEmpty())
     return null;
   return header.next.object;
 }
 public object last(){
   if (isEmpty())
         return null;
   return trailer.prev.object;
```

```
 }

public dnode  insertAfter(dnode p, Object element){
//Verilen bir düğümün sonrasına eleman yerleştirme
        numElts++;
        dnode newNode = new dnode(element, p,p.next);
        p.next=newNode;
        newNode.next.prev=newNode;
        return newNode;
}
public  dnode insertBefore(dnode p, Object element) {
//Verilen bir düğümün öncesine eleman yerleştirme
        numElms++;
        dnode newDNode = new dnode(element, p.prev, p);
        p.prev=newNode;
        newNode.prev.next=newNode;
        return newNode;
 }
public dnode insertFirst(Object element) {
//Başa eleman yerleştirme
        numElms++;
        dnode newNode = new dnode(element, header, header.next);
        header.next.prev=newNode;
        header.next=newNode;
        return newNode;
 }
public dnode insertLast(Object element) {
//Sona eleman yerleştirme
        numElms++;
        dnode newNode = new dnode(element, trailer.prev, trailer);
        trailer.prev=newNode;
        newNode.prev.next=newNode;
        return newNode;
 }
public Object delete(dnode p){
//Verilen düğümü silme
        numElms--;
        p.prev.next=p.next;
        p.next.prev=p.prev;
        p.prev=null;
        p.next=null;
        return p.element;
 }
```

En başa ve en sona eleman yerleştirme metodları insertBefore ve insertAfter medotları kullanılarak da yapılabilir.
Sone eleman eleman yerleştirme;

insertBefore(trailer,element);

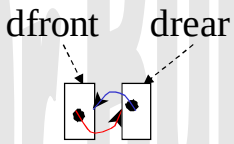
Başa eleman yerleştirme;

insertAfter(header,elements);

4.2 Çift Yönlü Kuyruk

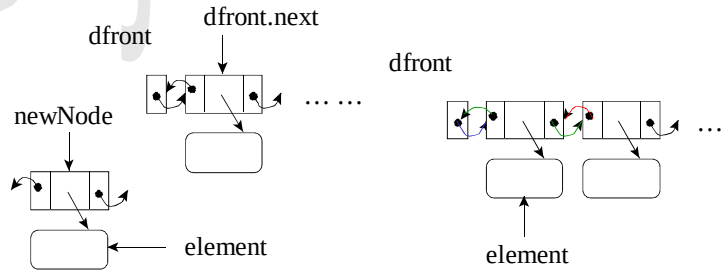
Yukarıda çift yönlü bağlı liste ve metodlarını verdikten sonra uygulama alanı olarak kuyruk yapısını gerçekleştirmek çok kolay olacaktır. Kuyruk yapısında alışılışa gelmiş olduğu için baş ve son kelimeleri için kullanılan front ve rear kelimeleri kullanılacaktır. Header kelimesi yerine front trailer kelimesi yerine rear kelimesi yazarak ve çift yönlü bağlı liste yapısı için verilen metodlardan uygun olanları kullanılarak çift yönlü kuyruk sınıfı aşağıdaki gibi tanımlana bilir.

Burada yine boş kuyrukta iki düğüm olacak ve bunlar birbirlerini gösterecektir. Şelil 4.6'da boş bir kuyruk gösterilmiştir.

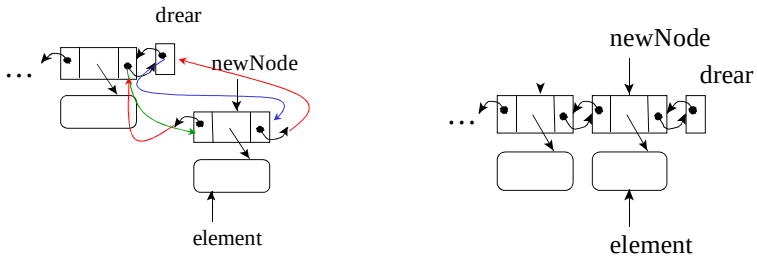


Şekil 4.6. Boş bir çift yönlü kuyruk.

Çift yönlü kuyruk yapısında tek yönlü de sunulan FIFO (İlk Giren İlk Çıkar) yapısının haricinde kuyruk mantıkları gerçekleştirmek oldukça rahattır. Eğer öncelikli bir düğüm varsa bu liste başına eklenerek öncelik tanıması mümkündür. Şekil 6.7'de baştan ve sondan eleman eklenmesi şematik olarak verilmiştir.



a)



b)

Şekil 4.7. Çift yönlü kuyruğa eleman eklenmesi, a) Baştan ekleme, b) Sondan ekleme.

Bu mantık içerisinde çift yönlü kuyruk mantığı java kodlaması olarak aşağıdaki gibi verilebilir.

```
public class dlQueue{
        dnode dfront, drear;
        int size;
        public dlQueue(){
                dfront=new dnode();
                drear=new dnode();
                dfront.next=drear;
                drear.prev.dfront;
                size=0;
        //Diğer Metodlar
        ….
        ….
        }

public dnode insertFront(Object element) {
        size++;
        dnode newNode = new dnode(element, dfront, dfront.next);
        dfront.next.prev=newNode;
        dfront.next=newNode;
        return newNode;
 }

public dnode insertRear(Object element) {
        size++;
        dnode newNode = new dnode(element, rear.prev, rear);
        rear.prev=newNode;
        newNode.prev.next=newNode;
        return newNode;
 }

public Object first(){
        if (isEmpty())
                return null;
                return dfront.next.element;
}
public Object last(){
        if( isEmpty() )
                return null;
                return drear.prev.element;
        }
public int size(){
        return size;
}
public boolean isEmpty( ){
        return dfront.next == drear;
}
public Object deleteFront() {
        if( isEmpty() )
                return null;
        dnode p=dfront.next;
        Object o = p.element;
```

```
        p.prev.next=p.next;,
        p.next.prev=p.prev;
        p.prev=null;
        p.next.null;
        size--;
        return o;
        }
}
public class GeneratedlQueue {
        public static void main(String args[]) {
                dlQueue D = new dlQueue();
                for (int i = 0; i < 10; i++) {
                        D.insertRear(new Integer(i));}
                for (i = 0; i < 10; i++)
                        System.out.print(((Integer) D.deleteFront()).intValue());
         }
}
```