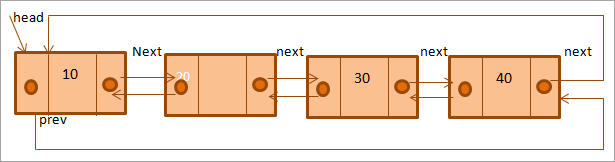
**DAİRESEL ÇİFT YÖNLÜ BAĞLI LİSTE**

Dairesel çift bağlantılı liste karmaşık yapılardan biridir. Bu listede, çift bağlantılı listenin son düğümü ilk düğümün adresini, ilk düğüm ise son düğümün adresini içerir.

Böylece dairesel bir çift bağlantılı listede bir döngü vardır ve düğüm işaretçilerinin hiçbiri null olarak ayarlanmamıştır.

Listenin ilk düğümü, önceki işaretçisindeki son düğümün adresini de içerir. Dairesel bir çift bağlantılı listenin yapısında üç parça içermesi nedeniyle, düğüm başına daha fazla alan ve daha pahalı temel işlemler gerektirir.

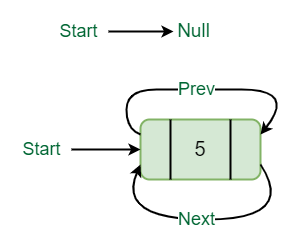
Bununla birlikte, dairesel bir çift bağlantılı liste, işaretçilerin kolayca değiştirilmesini sağlar ve arama, iki kat daha verimli hale gelir.



**Dairesel Çift Bağlantılı Listeye Ekleme:**

**1. Listenin sonuna veya boş bir listeye ekleme:**

**data = 5** olan bir düğüm(Say **N** ) eklenir . Bu nedenle, N'nin önceki işaretçisi N'yi ve sonraki N'nin işaretçisi de N'yi işaret eder. Ancak şimdi başlangıç ​​işaretçisi listenin ilk düğümünü işaret eder.



**2. Liste başlangıçta bazı düğümler içerir, Liste'nin ilk düğümüne başlangıç ​​noktaları:**

Veri = 7 olan bir düğüm (M Deyin) eklenir, bu nedenle M'nin önceki işaretçisi son düğümü gösterir, M'nin sonraki işaretçisi ilk düğümü ve son düğümün bir sonraki işaretçisi bu M düğümünü gösterir ve ilk düğümün önceki işaretçi bu M düğümüne işaret eder.

// Liste boşsa, tek bir liste oluşturun

// düğüm dairesel ve çift liste oluşur

if (start == null) {

Node new\_node = new Node();

new\_node.data = value;

new\_node.next = new\_node.prev = new\_node;

start = new\_node;

return;

// Liste boş değilse

// Son düğümü bul

Node last = (start).prev;

// Düğümü dinamik olarak oluştur

Node new\_node = new Node();

new\_node.data = value;

// Başlangıç olacak

// new\_node'un yanında

new\_node.next = start;

// Yeni düğümü başlangıçtan önce yap

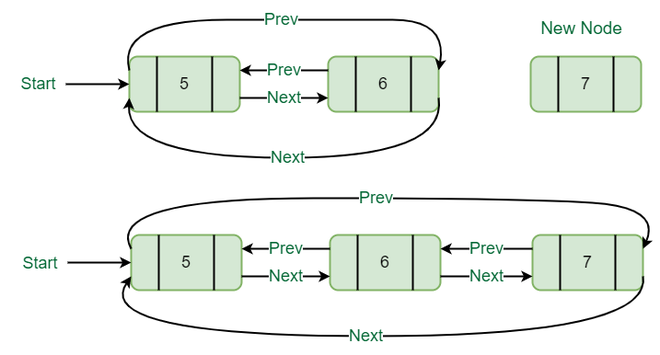
(start).prev = new\_node;

// Yeni düğümün sonuncusunu yap

new\_node.prev = last;

// Eski düğümün yanına yeni düğüm yap

last.next = yeni\_düğüm;



**3. Listenin başına ekleme:**

Listenin başına bir düğüm eklemek için, data = 5 olan bir düğüm oluşturun (T Deyin), T sonraki işaretçi listenin ilk düğümünü gösterir, T önceki işaretçi listenin son düğümünü gösterir, son düğümün sonrakini gösterir.

// Başa Düğüm ekleme işlevi

// Listenin,static void insertBegin(int value)

{

// İşaretçi son Düğüme işaret ediyor

Node last = (start).prev;

Node new\_node = new Node();

new\_node.data = value; // Inserting the data

// yeni düğümün önceki ve sonraki kurulumu

new\_node.next = start;

new\_node.prev = last;

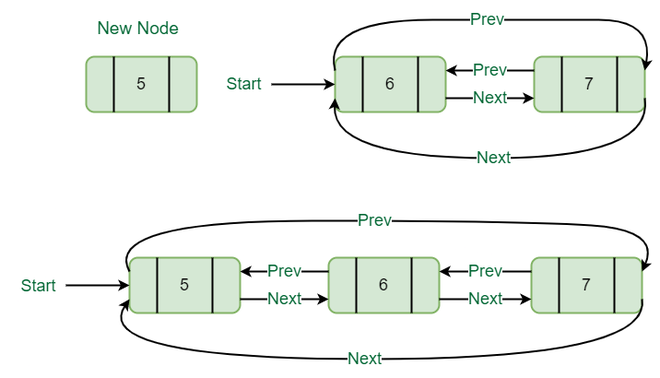
// Sonraki ve önceki başlangıç işaretçilerini güncelle son olarak

last.next = (start).prev = new\_node;

// Başlangıç işaretçisini güncelle lütfen

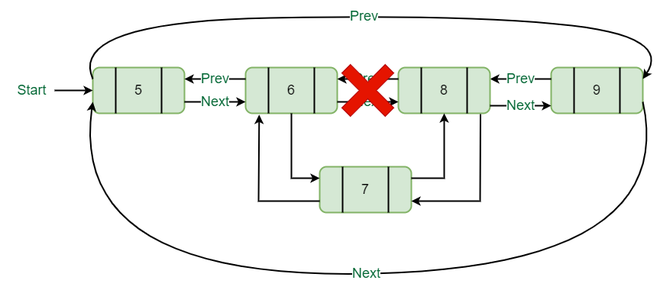
start = new\_node;

}



**4. Listenin düğümleri arasına ekleme:**

Liste arasına bir düğüm eklemek için, iki veri değeri gereklidir, bu değerden sonra yeni düğüm eklenir ve diğeri yeni düğümün verileridir.



// değeri değer1 olan düğüm ekleme işlevi olarak kullanılır.

// Yeni düğüm, düğümden sonra eklenir.

// değer2 ilestatic void insertAfter(int value1, int value2)

{

Node new\_node = new Node();

new\_node.data = value1; // Verilerin eklenmesi

// value2'ye sahip düğümü ve ondan sonraki düğümü bulun

Node temp = start;

while (temp.data != value2)

temp = temp.next;

Node next = temp.next;

// temp ve next arasına new\_node eklenir burada.

temp.next = new\_node;

new\_node.prev = temp;

new\_node.next = next;

next.prev = new\_node;

}

**Dairesel Çift Yönlü Bağlı Liste Silme Metodları**

// Belirli bir düğümü listeden silme işlevi

static Node deleteNode(Node start, int key)

{

// Liste boşsa

if (start == null)

return null;

// Gerekli düğümü bulun

// İki işaretçi bildir ve başlat

Node curr = start, prev\_1 = null;

while (curr.data != key) {

// Listede düğüm yoksa

if (curr.next == start) {

System.out.printf(key);

return start;

}

prev\_1 = curr;

curr = curr.next;

}

// Listedeki tek düğümün düğüm olup olmadığını kontrol edin

if (curr.next == start && prev\_1 == null) {

(start) = null;

return start;

}

// Liste elemanlarını gösteren fonksiyon

static void display(Node start)

{

Node temp = start;

while (temp.next != start) {

System.out.printf("%d ", temp.data);

temp = temp.next;

}

System.out.printf("%d ", temp.data);

}

// Listede olmayan düğümü sil

start = deleteNode(start, 9);

System.out.printf("\nList After Deletion: ");

display(start);

**Avantajları**

1. Her düğümün önceki ve sonraki düğümlere işaret eden işaretçileri olduğundan, çift bağlantılı liste hem ileri hem de geri yönde kolayca geçilebilir.
2. Yalnızca işaretçileri değiştirerek yeni düğümü hızlı bir şekilde ekleyebilirsiniz.
3. Benzer şekilde, silme işlemi için önceki ve sonraki işaretçilere sahip olduğumuzdan, silme daha kolaydır ve tek bağlantılı liste durumunda olduğu gibi önceki düğümü bulmak için tüm listeyi geçmemize gerek yoktur.

**Dezavantajları**

1. Çift bağlantılı listede bir önceki işaretçi gibi fazladan bir işaretçi olduğundan, bu işaretçiyi bir sonraki işaretçi ve veri öğesiyle birlikte depolamak için ek bellek alanı gerekir.
2. Ekleme, silme vb. gibi tüm işlemler hem önceki hem de sonraki işaretçilerin manipüle edilmesini gerektirir, böylece operasyonel ek yük getirir.

ÖĞRENCİ

YILDIRAY ÖCAL

4.SINIF

VERİ YAPILARI (İKİNCİ ÖĞRETİM)

02200201096

[Yldry1906@gmail.com](mailto:Yldry1906@gmail.com)

Veri yapıları 4.ödev