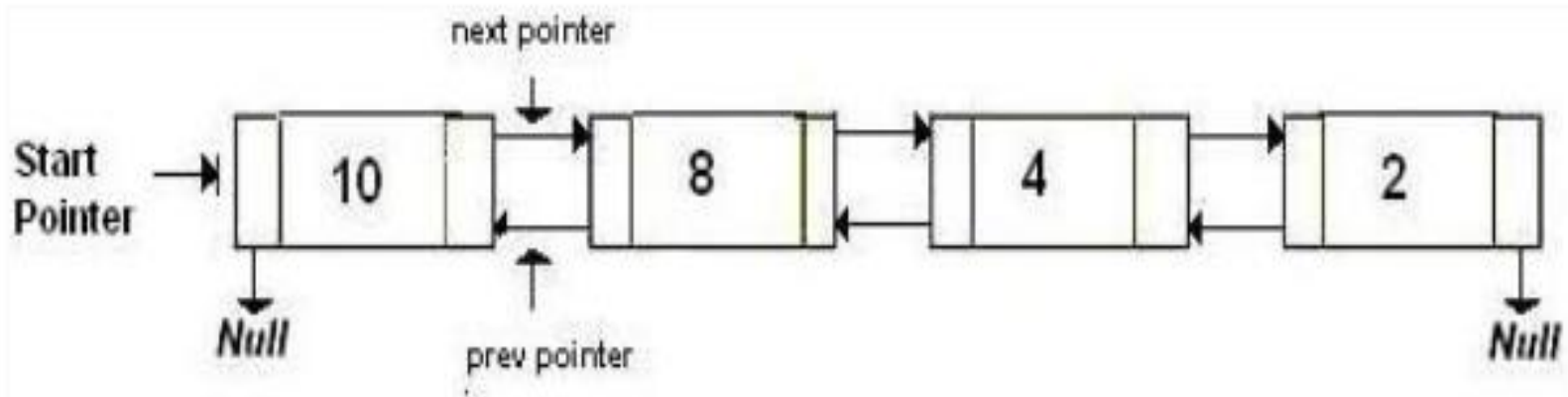


Pertemuan 5

LINKED LIST LANJUTAN

Doubly Linked List

Dalam single linked list, setiap node dari list memiliki dua komponen, nilai aktual dari node dan referensi ke node berikutnya dalam linked list. Dalam doubly linked list, setiap node memiliki tiga komponen: nilai node, referensi ke node sebelumnya, dan referensi ke node berikutnya. Untuk node awal dari daftar tertaut ganda, referensi ke node sebelumnya adalah null. Demikian pula, untuk node terakhir dalam daftar tertaut ganda, referensi ke node berikutnya adalah null.



Pembuatan Doubly Linked List

Membuat class untuk node

class Node:

```
def __init__(self, data):  
    self.item = data  
    self.nref = None  
    self.pref = None
```

Pada kode di atas, membuat class Node dengan tiga variabel anggota: item, nref, dan pref. Variabel item akan menyimpan data aktual untuk node. Nref menyimpan referensi ke node berikutnya, sementara pref menyimpan referensi ke node sebelumnya dalam doubly linked list. Selanjutnya, perlu membuat class DoublyLinkedList, yang berisi fungsi terkait doubly linked list yang berbeda.

class DoublyLinkedList:

```
def __init__(self):  
    self.start_node = None
```

Menambahkan Item pada Empty List

Memasukkan Item dalam Empty List Cara termudah untuk menyisipkan item dalam doubly linked list adalah dengan menyisipkan item ke dalam Empty List. Skrip berikut menyisipkan elemen di awal doubly linked list:

```
def insert_in_emptylist(self, data):  
    if self.start_node is None:  
        new_node = Node(data)  
        self.start_node = new_node  
    else:  
        print("list is not empty")
```

Definisikan metode `insert_in_emptylist ()`. Metode pertama memeriksa apakah variabel `self.start_node` adalah `None` atau tidak. Jika variabelnya Tidak Ada, itu berarti list kosong. Selanjutnya, node baru dibuat dan nilainya diinisialisasi oleh nilai yang diteruskan sebagai parameter ke parameter `data` dari fungsi `insert_in_emptylist ()`. Terakhir, nilai variabel `self.start_node` disetel ke node baru. Dalam kasus jika list tidak kosong, pesan hanya ditampilkan list tidak kosong.

Menambahkan Item di Awal Doubly Linked List

Untuk memasukkan item di awal doubly linked list, pertama harus memeriksa apakah list tersebut kosong atau tidak. Jika list kosong, definisikan di `insert_in_emptylist ()` untuk memasukkan elemen karena dalam list kosong, elemen pertama selalu di awal. Jika tidak, jika daftarnya tidak kosong, lakukan tiga operasi: Untuk node baru, referensi ke node berikutnya akan disetel ke `self.start_node`. Untuk `self.start_node` referensi ke node sebelumnya akan diatur ke node yang baru dimasukkan. Terakhir, `self.start_node` akan menjadi node yang baru disisipkan.

```
def insert_at_start(self, data):
```

```
    if self.start_node is None:
```

```
        new_node = Node(data)
```

```
        self.start_node = new_node
```

```
        print("node inserted")
```

```
    return
```

```
        new_node = Node(data)
```

```
        new_node.nref = self.start_node
```

```
        self.start_node.pref = new_node
```

```
        self.start_node = new_node
```

Menambahkan Item di Akhir Doubly Linked List

Memasukkan elemen di akhir doubly linked list agak mirip dengan memasukkan elemen di awal. Pertama, periksa apakah list kosong. Jika list kosong maka kita cukup menggunakan metode `insert_in_emptylist ()` untuk memasukkan elemen. Jika list sudah berisi beberapa elemen, kita akan menelusuri list tersebut hingga referensi ke node berikutnya menjadi Tidak Ada. Ketika referensi node berikutnya menjadi None itu berarti node saat ini adalah node terakhir. Referensi sebelumnya untuk node baru diatur ke node terakhir, dan referensi berikutnya untuk node terakhir diatur ke node yang baru dimasukkan.

```
def insert_at_end(self, data):  
    if self.start_node is None:  
        new_node = Node(data)  
        self.start_node = new_node  
        return  
    n = self.start_node  
    while  
        n.nref is not None:  
            n = n.nref  
    new_node = Node(data)  
    n.nref = new_node  
    new_node.pref = n
```

Menambahkan Item pada After Another Item

Untuk menyisipkan item setelah item lain, periksa apakah list kosong atau tidak. Jika list kosong, tampilkan pesan "list is empty". Jika tidak, lakukan iterasi melalui semua node dalam doubly linked list. Jika ingin menyisipkan node pada node setelahnya yang dituju tidak ditemukan, tampilkan pesan bahwa item tersebut tidak ditemukan. Lain jika node ditemukan, itu dipilih dan lakukan empat operasi: Tetapkan referensi sebelumnya dari node yang baru dimasukkan ke node yang dipilih. Tetapkan referensi berikutnya dari node yang baru disisipkan ke referensi berikutnya dari yang dipilih. Jika node yang dipilih bukan node terakhir, atur referensi sebelumnya dari node berikutnya setelah node yang dipilih ke node yang baru ditambahkan. Terakhir, atur referensi berikutnya dari node yang dipilih ke node yang baru dimasukkan.

```
def insert_after_item(self, x, data):  
    if self.start_node is None:  
        print("List is empty")  
        return  
    else: n = self.start_node  
    While  
        n is not None:  
            if n.item == x:  
                break  
            n = n.nref  
        if n is None:  
            print("item not in the list")  
        else:  
            new_node = Node(data)  
            new_node.pref = n  
            new_node.nref = n.nref  
            if n.nref is not None: n.nref.prev =  
                new_node n.nref = new_node
```


Menambahkan Item Before Another Item

Untuk menyisipkan item sebelum item lain, pertama periksa apakah list kosong atau tidak. Jika list kosong, tampilkan pesan bahwa "list is empty". Jika tidak, lakukan iterasi melalui semua node dalam doubly linked list. Jika node yang sebelumnya ingin memasukkan node baru tidak ditemukan, tampilkan pesan bahwa item tersebut tidak ditemukan. Jika node ditemukan, itu dipilih dan lakukan empat operasi:

1. Tetapkan referensi berikutnya dari node yang baru dimasukkan ke node yang dipilih.
2. Setel referensi sebelumnya dari node yang baru disisipkan ke referensi sebelumnya dari yang dipilih.
3. Tetapkan referensi berikutnya dari node sebelumnya ke node yang dipilih, ke node yang baru ditambahkan.
4. Terakhir, atur referensi sebelumnya dari node yang dipilih ke node yang baru dimasukkan.

Menambahkan Item Before Another Item (Lanjutan)

```
def insert_before_item(self, x, data):  
    if self.start_node is None:  
        print("List is empty")  
        return  
    else:  
        n = self.start_node  
        while n is not None:  
            if n.item == x:  
                break  
            n = n.nref if n is None:  
            print("item not in the list")  
        else:  
            new_node = Node(data)  
            new_node.nref = n  
            new_node.pref = n.pref  
            if n.pref is not None:  
                n.pref.nref = new_node  
            n.pref = new_nodeZ
```

Menghapus Elemen dari Doubly Linked List

Menghapus Elemen di node awal

Cara termudah untuk menghapus elemen dari doubly linked list adalah di awal. Untuk melakukannya, mengatur nilai dari simpul awal ke simpul berikutnya dan kemudian mengatur referensi sebelumnya dari simpul awal ke None. Namun sebelumnya lakukan dua pemeriksaan. Pertama, apakah list kosong. Dan kemudian lihat apakah daftar tersebut hanya berisi satu elemen atau tidak. Jika list hanya berisi satu elemen maka kita dapat mengatur node awal ke None

```
def delete_at_start(self):  
    if self.start_node is None:  
        print("The list has no element to delete")  
    Return  
    if self.start_node.nref is None:  
        self.start_node = None  
    return  
    self.start_node = self.start_node.nref  
    self.start_prev = None;
```

Menghapus Elemen di Node Akhir

Periksa kembali apakah list kosong atau jika list berisi satu elemen. Jika list berisi satu elemen, atur node awal ke None. Jika list memiliki lebih dari satu elemen, ulangi list sampai node terakhir tercapai. Setelah mencapai node terakhir, tetapkan referensi berikutnya dari node sebelumnya ke node terakhir, ke None yang benar-benar menghapus node terakhir.

```
def delete_at_end(self):  
if self.start_node is None:  
print("The list has no element to delete")  
Return  
if self.start_node.nref is None:  
self.start_node = None  
Return  
n = self.start_node  
while n.nref is not None:  
n = n.nref  
n.nref.nref = None
```

Menghapus Elemen Berdasarkan Nilai

```
def delete_element_by_value(self, x):  
    if self.start_node is None:  
        print("The list has no element to delete")  
        Return  
        if self.start_node.nref is None:  
            if self.start_node.item == x:  
                self.start_node = None  
            else:  
                print("Item not found")  
        Return  
        if self.start_node.item == x:  
            self.start_node = self.start_node.nref  
            self.start_node.pref = None
```

```
        return  
        n = self.start_node  
        while n.nref is not None:  
            if n.item == x:  
                break;  
            n = n.nref  
            if n.nref is not None: n.pref.nref  
            = n.nref  
            n.nref.pref = n.pref  
        else:  
            if n.item == x:  
                n.pref.nref = None  
            else:  
                print("Element not found")
```

Implementasi Doubly Linked List

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None
class DoubleLinkedList:
    def __init__(self):
        self.head = None
    def append(self, data):
        if self.head is None:
            new_node = Node(data)
            new_node.prev = None
            self.head = new_node
        else:
            new_node = Node(data)
            cur = self.head
            while cur.next:
```

```
        cur.next = new_node
        new_node.prev = cur
        new_node.next = None
    def prepend(self, data):
        if self.head is None:
            new_node = Node(data)
            new_node.next = self.head
            self.head = new_node
        else:
            new_node = Node(data)
            self.head.prev = new_node
            new_node.next = self.head
            self.head = new_node
            new_node.prev = None
    def print_list(self):
        cur = self.head
        while cur:
            print(cur.data)
            cur = cur.next
```

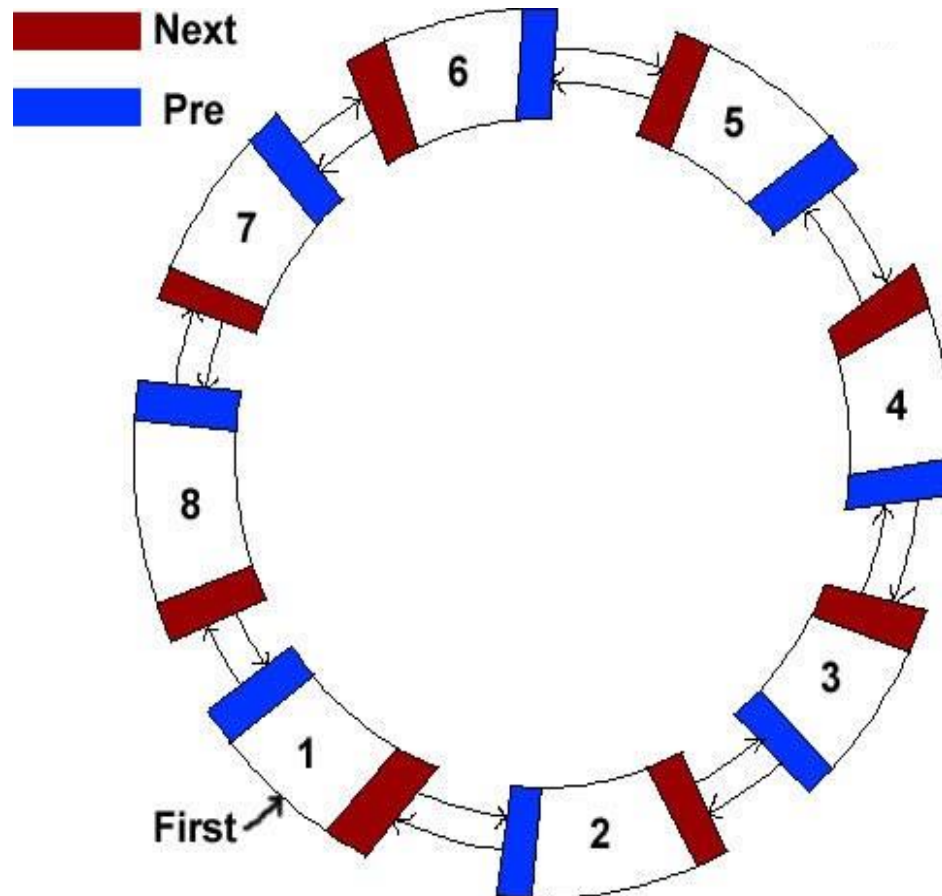
Implementasi Doubly Linked List (Lanjutan)

```
d1 = int(input("Masukkan nilai : "))  
d2 = int(input("Masukkan nilai : "))  
d3 = int(input("Masukkan nilai : "))  
d4 = int(input("Masukkan nilai : "))  
d5 = int(input("Masukkan nilai : "))  
d6 = int(input("Masukkan nilai : "))  
e1 = int(input("Masukkan nilai : "))  
e2 = int(input("Masukkan nilai : "))  
list=DoubleLinkedList()  
list.append(d1)  
list.append(d2)  
list.append(d3)
```

```
list.append(d4)  
list.append(d5)  
list.append(d6)  
list.prepend(e1)  
list.prepend(e2)  
list.print_list()
```

Circular Linked List

Circular Linked List memiliki 2 pointer pada masing - masing node, dan pada pointer previous pada node tail mengarah ke pointer next pada node head



Circular Linked List

```
class Node:
```

```
def __init__(self, data):
```

```
self.data = data
```

```
self.next = None
```

```
class CircularLinkedList:
```

```
def __init__(self):
```

```
self.head = None
```

```
def prepend(self, data):
```

```
new_node = Node(data)
```

```
cur = self.head
```

```
new_node.next = self.head
```

```
if not self.head:
```

```
new_node.next = new_node
```

```
else:
```

```
while cur.next != self.head:
```

```
cur = cur.next
```

```
cur.next = new_node
```

```
self.head = new_node
```

```
def append(self, data):
```

```
if not self.head:
```

```
self.head = Node(data)
```

```
self.head.next = self.head
```

```
else:
```

```
new_node = Node(data)
```

```
cur = self.head
```

```
while cur.next != self.head:
```

```
cur = cur.next
```

```
cur.next = new_node
```

```
new_node.next = self.head
```

```
def print_list(self):
```

```
cur = self.head
```

```
    while cur:
```

```
        print(cur.data)
```

```
        cur = cur.next
```

```
        if cur == self.head:
```

```
            break
```

```
c1 = int(input("Masukkan Nilai : "))
```

```
c2 = int(input("Masukkan Nilai : "))
```

```
f1 = int(input("Masukkan Nilai : "))
```

```
f2 = int(input("Masukkan Nilai : "))
```

```
list = CircularLinkedList()
```

```
list.append(c1)
```

```
list.append(c2)
```

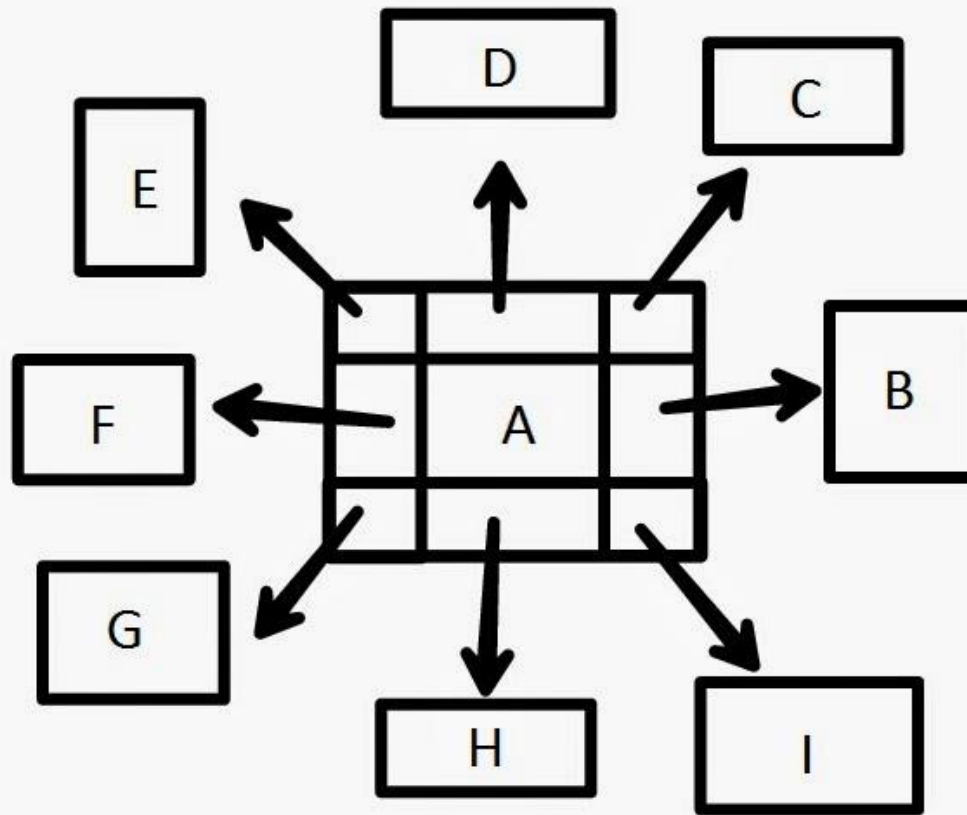
```
list.prepend(f1)
```

```
list.prepend(f2)
```

```
list.print_list()
```

Multiple Linked List

Multiple Linked List adalah Linked List yang dimana pada setiap node memiliki banyak pointer.



Latihan Soal Struktur Data

(Pertemuan 5)

1. Dalam single linked list, setiap node dari list memiliki dua komponen, nilai aktual dari node dan referensi ke node berikutnya dalam linked list merupakan pengertian...

- a. Dinamic Variabel
- b. Pointer
- c. Double Linked List
- d. Single Linked
- e. Circular

2. if self.start_node is None:

Maksud dari perintah tersebut adalah....

- a. Keadaan node sedang ditunjuk oleh pointer
- b. Memeriksa apakah variabel self.start_node adalah None atau tidak
- c. Harus menunjuk ke node berikutnya agar list tidak putus
- d. Setelah head lama akan menjadi head baru
- e. Jika head masih NULL

2. if self.start_node is None:

Maksud dari perintah tersebut adalah....

- a. Keadaan node sedang ditunjuk oleh pointer
- b. Memeriksa apakah variabel self.start_node adalah None atau tidak
- c. Harus menunjuk ke node berikutnya agar list tidak putus
- d. Setelah head lama akan menjadi head baru
- e. Jika head masih NULL

3. Linked List yang dimana pada setiap node memiliki banyak pointer adalah pengertian dari ...

- | | |
|-----------------------|-------------------------|
| a. Single Linked List | d. Multiple Linked List |
| b. Doubly Linked List | e. Join Linked List |
| c. Many Linked List | |

3. Linked List yang dimana pada setiap node memiliki banyak pointer adalah pengertian dari ...
- a. Single Linked List
 - b. Doubly Linked List
 - c. Many Linked List
 - d. Multiple Linked List
 - e. Join Linked List
4. **insert_in_emptylist ()** adalah perintah untuk....
- a. Menambahkan elemen
 - b. Mengosongkan elemen
 - c. Menyisipkan elemen
 - d. Menambah di tengah
 - e. Meghapus elemen

4. **insert_in_emptylist ()** adalah perintah untuk....
- a. Menambahkan elemen
 - b. Mengosongkan elemen
 - c. Menyisipkan elemen
 - d. Menambah di tengah
 - e. Meghapus elemen
5. Berikut adalah metode penghapusan elemen pada Doubly Linked List...
- a. Menghapus berdasarkan nama
 - b. Menghapus berdasarkan nilai
 - c. Menghapus dengan cepat
 - d. Menghapus di tengah
 - e. Menghapus elemen genap

5. Berikut adalah metode penghapusan elemen pada Doubly Linked List...
 - a. Menghapus berdasarkan nama
 - b. Menghapus berdasarkan nilai
 - c. Menghapus dengan cepat
 - d. Menghapus di tengah
 - e. Menghapus elemen genap

1. Dalam single linked list, setiap node dari list memiliki dua komponen, nilai aktual dari node dan referensi ke node berikutnya dalam linked list merupakan pengertian...
 - a. Dinamic Variabel
 - b. Pointer
 - c. Double Linked List
 - d. Single Linked
 - e. Circular