

LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL KE-2

LINKEDLIST PADA PYTHON



Disusun Oleh:

Nama : Oktario Mufti Yudha
NPM : 2320506044
Kelas : 04 (Empat)

Program Studi S1 Teknologi Informasi

Fakultas Teknik, Universitas Tidar

Genap 2023/2024

I. Tujuan Praktikum

Praktikum ini bertujuan untuk memperkenalkan mahasiswa pada konsep dasar struktur data linked list serta memberikan pemahaman praktis dalam mengimplementasikannya menggunakan bahasa pemrograman Python. Dengan mempelajari praktikum ini, mahasiswa diharapkan dapat memahami prinsip-prinsip dasar dalam pembuatan, penyisipan, dan penghapusan node dalam linked list. Selain itu, praktikum ini juga dimaksudkan untuk membantu mahasiswa memahami bagaimana linked list bekerja secara internal dan bagaimana mereka dapat memanfaatkannya dalam pembangunan aplikasi yang lebih kompleks. Melalui serangkaian latihan praktis, diharapkan mahasiswa dapat menguasai konsep dasar ini dan mampu menerapkannya dalam konteks pemrograman nyata.

II. Dasar Teori

Linked list merupakan struktur data yang terdiri dari serangkaian node. Tiap node memiliki dua bagian yang berisi data dan referensi ke node berikutnya. Ada dua jenis linked list yaitu linked list satu arah dan linked list dua arah.

Operasi dasar yang biasa digunakan adalah:

1. insertion : untuk memasukkan elemen baru ke dalam linked list
2. deletion : untuk menghapus element dari linked list.
3. search : untuk mencari element tertentu dalam sebuah linked list.

Tanda Tangan

III. Hasil dan Pembahasan

a. Membuat Node

```
#Membuat Node
class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None

n1 = Node('Januari')
n2 = Node('Februari')
n3 = Node('Maret')

print(n1.dataval)
print(n2.dataval)
print(n3.dataval)
print('=====Add=====')
print()
```

Gambar 3.1: Membuat Node

1. `class Node:` membuat kelas Python yang bernama Node. Kelas ini akan digunakan untuk merepresentasikan node dalam linked list.
2. `def __init__(self, dataval=None):` konstruktor kelas Node. akan dipanggil ketika objek Node baru dibuat.
3. `self.dataval = dataval:` Membuat variabel `dataval` yang akan menyimpan data yang diberikan kepada node saat objek dibuat.
4. `self.nextval = None:` Membuat variabel instance `nextval` yang akan digunakan untuk menunjukkan node berikutnya dalam linked list.
5. `n1 = Node('Januari'):` Membuat objek `n1` dari kelas Node dengan data 'Januari'.
6. `n2 = Node('Februari'):` Membuat objek `n2` dari kelas Node dengan data 'Februari'.
7. `n3 = Node('Maret'):` Membuat objek `n3` dari kelas Node dengan data 'Maret'.
8. `print(n1.dataval):` Mencetak nilai `dataval` dari objek `n1`, yang seharusnya mencetak 'Januari'.
9. `print(n2.dataval):` Mencetak nilai `dataval` dari objek `n2`, yang seharusnya mencetak 'Februari'.

Tanda Tangan

10. `print(n3.dataval)`: Mencetak nilai `dataval` dari objek `n3`, yang seharusnya mencetak 'Maret'.
11. `print('=====Add=====')`: Mencetak garis pemisah untuk membedakan output.
12. `print()`: Mencetak baris kosong.

b. Membuat class linked list

```
#Membuat Class Linked List
class LinkedList:
    def __init__(self):
        self.headval = None
```

Gambar 3.2: Class Linked List

1. `class LinkedList`: Ini mendeklarasi kelas `LinkedList`
2. `def __init__(self)`: Yang akan dipanggil saat objek `LinkedList` baru dibuat.
3. `self.headval = None`: Membuat variabel instance `headval` yang akan digunakan untuk menunjukkan head dari linked list.

c. Fungsi menampilkan linked list

```
#Menampilkan Linked List
def listprint(self):
    printval = self.headval
    while printval is not None:
        print(printval.dataval)
        printval = printval.nextval
```

Gambar 3.3: Menampilkan Linked List

1. `def listprint(self)`: Ini adalah fungsi yang digunakan untuk mencetak semua nilai dalam linked list.
2. `printval = self.headval`: Membuat variabel lokal `printval` dan menginisiasinya dengan head dari linked list.
3. `while printval is not None`: Memulai loop while yang akan terus berjalan selama `printval` tidak `None`.

Tanda Tangan

4. `print(printval.dataval)`: Mencetak nilai `dataval` dari node yang sedang dikunjungi.
5. `printval = printval.nextval`: Menggerakkan `printval` ke node berikutnya dalam linked list dengan mengatur `printval` sama dengan `nextval` dari node saat ini.

d. Penyisipan di awal

```
#Penyisipan di Awal
def AtBegining(self, newdata):
    NewNode = Node(newdata)
    NewNode.nextval = self.headval
    self.headval = NewNode
```

Gambr 3.4: Penyisipan di Awal

1. `def AtBegining(self, newdata)::` Ini adalah fungsi yang digunakan untuk menambahkan sebuah node baru di awal linked list.
2. `NewNode = Node(newdata)`: Membuat sebuah node baru dengan menggunakan kelas `Node` dan melewati `newdata` sebagai argumen untuk data node baru ini.
3. `NewNode.nextval = self.headval`: Mengatur `nextval` dari node baru (`NewNode`) menjadi head dari linked list saat ini.
4. `self.headval = NewNode`: Mengatur head dari linked list menjadi node baru (`NewNode`).

e. Penyisipan di Tengah

```
#Penyisipan di Tengah
def AddInBetween(self, mid_node, newdata):
    if mid_node is None:
        print("The mentioned node is absent")
        return
    NewNode = Node(newdata)
    NewNode.nextval = mid_node.nextval
    mid_node.nextval = NewNode
```

Gambar 3.5: Penyisipan di Tengah

Tanda Tangan

1. `def AddInBetween(self, mid_node, newdata)::` Ini adalah fungsi yang digunakan untuk menambahkan sebuah node baru di antara dua node yang sudah ada dalam linked list.
2. `if mid_node is None:` Memeriksa apakah `mid_node` (node di antara yang baru akan ditambahkan) tidak ada.
3. `print("The mentioned node is absent"):` Mencetak pesan bahwa node yang ditentukan tidak ada dalam linked list.
4. `return:` Mengembalikan eksekusi metode karena tidak mungkin menambahkan node baru jika node di antara yang ditentukan tidak ada.
5. `NewNode = Node(newdata):` Membuat sebuah node baru dengan menggunakan kelas `Node` dan melewati `newdata` sebagai argumen untuk data node baru ini.
6. `NewNode.nextval = mid_node.nextval:` Mengatur `nextval` dari node baru (`NewNode`) menjadi `nextval` dari `mid_node`.
7. `mid_node.nextval = NewNode:` Mengatur `nextval` dari `mid_node` menjadi node baru (`NewNode`).

f. Penyisipan di Akhir

```
#Penyisipan di Akhir
def AddInEnd(self, newdata):
    NewNode = Node(newdata)
    if self.headval is None:
        self.headval = NewNode
        return
    last = self.headval
    while(last.nextval):
        last = last.nextval
    last.nextval = NewNode
```

Gambar 3.6: Penyisipan di Akhir

1. `def AddInEnd(self, newdata):` Ini adalah fungsi yang digunakan untuk menambahkan sebuah node baru di akhir linked list.

Tanda Tangan

2. `NewNode = Node(newdata)`: Membuat sebuah node baru dengan menggunakan kelas `Node` dan melewati `newdata` sebagai argumen untuk data node baru ini.
3. `if self.headval is None`: Memeriksa apakah linked list kosong.
4. `self.headval = NewNode`: Jika linked list kosong, maka node baru (`NewNode`) akan menjadi kepala (`head`) dari linked list.
5. `return`: Mengembalikan eksekusi metode.
6. `last = self.headval`: Menginisialisasi variabel `last` dengan `head` dari linked list.
7. `while(last.nextval)`: Memulai loop `while` yang akan terus berjalan selama `last.nextval` tidak `None`, yang berarti `last` bukan merupakan node terakhir dalam linked list.
8. `last = last.nextval`: Memindahkan `last` ke node berikutnya dalam linked list dengan mengatur `last` sama dengan `nextval` dari node saat ini.
9. `last.nextval = NewNode`: Ketika loop selesai, `last` akan menunjuk ke node terakhir dalam linked list.

g. Menghapus node dengan kunci

```
#Menghapus Node dengan Kunci
def RemoveNode(self, Removekey):
    HeadVal = self.headval
    if (HeadVal is not None):
        if (HeadVal.dataval == Removekey):
            self.headval = HeadVal.nextval
            HeadVal = None
            return
        while (HeadVal is not None):
            if HeadVal.dataval == Removekey:
                break
            prev = HeadVal
            HeadVal = HeadVal.nextval
        if (HeadVal == None):
            return
        prev.nextval = HeadVal.nextval
        HeadVal = None
```

Gambar 3.7: Menghapus Node dengan Kunci

1. `def RemoveNode(self, Removekey)`: Ini adalah fungsi yang digunakan untuk menghapus node dari linked list berdasarkan kunci yang diberikan.
2. `HeadVal = self.headval`: Membuat salinan dari `head` linked list dan menyimpannya dalam variabel `HeadVal`.

Tanda Tangan

3. if (HeadVal is not None): Memeriksa apakah linked list tidak kosong.
4. if (HeadVal.dataval == Removekey): Memeriksa apakah nilai dari node yang akan dihapus adalah sama dengan nilai yang diberikan.
5. self.headval = HeadVal.nextval: Jika node yang akan dihapus adalah head dari linked list, maka head linked list diperbarui ke node berikutnya, sehingga node pertama dihapus dari linked list.
6. HeadVal = None: Menetapkan HeadVal ke None untuk menghapus referensi ke node yang akan dihapus.
7. return: Mengembalikan eksekusi metode karena node yang akan dihapus telah ditemukan dan dihapus.
8. while (HeadVal is not None): Memulai loop while yang akan terus berjalan selama HeadVal tidak None.
9. if HeadVal.dataval == Removekey: Memeriksa apakah nilai dari node yang sedang diperiksa adalah sama dengan nilai yang diberikan.
10. prev = HeadVal: Menyimpan node sebelumnya dalam variabel prev
11. HeadVal = HeadVal.nextval: Memindahkan HeadVal ke node berikutnya dalam linked list dengan mengatur HeadVal sama dengan nextval dari node saat ini.
12. if (HeadVal == None): Memeriksa apakah HeadVal adalah None, yang menandakan bahwa pencarian telah mencapai akhir linked list tanpa menemukan node yang akan dihapus.
13. return: Mengembalikan eksekusi metode karena node yang akan dihapus tidak ditemukan dalam linked list.
14. prev.nextval = HeadVal.nextval: Menghubungkan node sebelum node yang akan dihapus dengan node setelahnya.
15. HeadVal = None: Menetapkan HeadVal ke None untuk menghapus referensi ke node yang akan dihapus.

Tanda Tangan

h. Menghapus node di awal

```
#Menghapus Node di Awal
def RemoveFirst(self):
    afterhead = self.headval
    self.headval = afterhead.nextval
```

Gambar 3.8: Menghapus Node di Awal

1. `def RemoveFirst(self)::` Ini adalah fungsi yang digunakan untuk menghapus node pertama dari linked list.
2. `afterhead = self.headval:` Membuat salinan dari node setelah head linked list dan menyimpannya dalam variabel `afterhead`.
3. `self.headval = afterhead.nextval:` Mengatur `headval` dari linked list ke `nextval` dari node setelah head linked list.

i. Menghapus node di akhir

```
#Menghapus Node di Akhir
def RemoveEnd(self):
    last = self.headval
    while(last is not None):
        if last.nextval == None:
            break
        prev = last
        last = last.nextval
    if (last == None):
        return
    prev.nextval = last.nextval
    last = None
```

Gambar 3.9: Menghapus Node di Akhir

1. `def RemoveEnd(self):` fungsi yang digunakan untuk menghapus node terakhir dari linked list.
2. `last = self.headval:` Membuat salinan dari head linked list dan menyimpannya dalam variabel `last`.
3. `while(last is not None):` Memulai loop while yang akan terus berjalan selama `last` tidak `None`.
4. `if last.nextval == None:` Memeriksa apakah node saat ini (`last`) adalah node terakhir dalam linked list.

Tanda Tangan

5. break: Jika node saat ini adalah node terakhir dalam linked list, loop berhenti.
6. prev = last: Menyimpan node sebelumnya dalam variabel prev
7. last = last.nextval: Memindahkan last ke node berikutnya dalam linked list dengan mengatur last sama dengan nextval dari node saat ini.
8. if (last == None): Memeriksa apakah last adalah None, yang menandakan bahwa linked list kosong.
9. return: Mengembalikan eksekusi metode.
10. prev.nextval = last.nextval: Menghubungkan node sebelum node yang akan dihapus dengan node setelahnya.
11. last = None: Menetapkan last ke None untuk menghapus referensi ke node yang dihapus.

j. Menjalankan fungsi-fungsi yang sudah di buat

```

Li = LinkedList()
Li.headval = n1
Li.headval.nextval = n2
n2.nextval = n3

Li.AtBegining('===Start===')
Li.AddInBetween(n1, '===Middle===')
Li.AddInEnd('===The Last===')
Li.listprint()
print()

print('=====Remove=====')
Li.RemoveNode('Maret')
Li.listprint()
print()

print('=====Remove First=====')
Li.RemoveFirst()
Li.listprint()
print()

print('=====Remove Last=====')
Li.RemoveEnd()
Li.listprint()

```

Gambar 3.10: Menjalankan Fungsi-Fungsi Yang Sudah di Buat

Pada tahap awal, linked list terdiri dari node-node '===Start===', 'Februari', dan '===The Last===', yang telah ditambahkan di awal, di antara dua node, dan di akhir linked list. Setelah operasi penghapusan dilakukan,

Tanda Tangan

node 'Maret' dihapus dari linked list. Kemudian, node pertama ('===Start===') dihapus dengan menggunakan metode RemoveFirst(). Terakhir, node terakhir ('===The Last===') dihapus dengan menggunakan metode RemoveEnd().

k. Output

```
Januari
Februari
Maret
=====Add=====

===Start===
Januari
===Middle===
Februari
Maret
===The Last===

=====Remove=====
===Start===
Januari
===Middle===
Februari
===The Last===

=====Remove First=====
Januari
===Middle===
Februari
===The Last===

=====Remove Last=====
Januari
===Middle===
Februari
```

Gambar 3.11: Output

Setelah menulis code diatas setelah di running maka akan menampilkan output seperti gambar diatas.

Tanda Tangan

IV. Latihan

a. Latihan 1

```
#Latihan 1
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None
        self.prev = None

n1 = Node('Cumi')
n2 = Node('Kerapu')
n3 = Node('Lele')

print(n1.data)
print(n2.data)
print(n3.data)
print()

class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def printList(self):
        curNode = self.head
        while curNode:
            print(curNode.data)
            curNode = curNode.next

    def addInBetween(self, key, data):
        curNode = self.head
        while curNode:
            if curNode.next is None and curNode.data == key:
                self.append(data)
                return
            elif curNode.data == key:
                newNode = Node(data)
                nxt = curNode.next
                curNode.next = newNode
                newNode.next = nxt
                newNode.prev = curNode
                nxt.prev = newNode
                curNode = curNode.next

def deleteNode(self, key):
    curNode = self.head
    while curNode:
        if curNode.data == key and curNode == self.head:
            if not curNode.next:
                curNode = None
                self.head = None
                return
            else:
                nxt = curNode.next
                curNode.next = None
                nxt.prev = None
                curNode = None
                self.head = nxt
                return
        elif curNode.data == key:
            if curNode.next:
                nxt = curNode.next
                prev = curNode.prev
                prev.next = nxt
                nxt.prev = prev
                curNode.next = None
                curNode.prev = None
                curNode = None
                return
            else:
                prev = curNode.prev
                prev.next = None
                curNode.prev = None
                curNode = None
                return
            curNode = curNode.next

li = DoublyLinkedList()
li.head = n1
n1.next = n2
n2.prev = n1
n2.next = n3
n3.prev = n2

print('=====Add in Between=====')
li.addInBetween('Kerapu', 'Bawal')
li.printList()
print()

print('=====Delete Node=====')
li.deleteNode('Kerapu')
li.printList()
```

Gambar 4.1: Latihan 1

Kode di atas mendefinisikan dua kelas, yaitu Node dan DoublyLinkedList, untuk membangun sebuah doubly linked list. Kelas Node digunakan untuk merepresentasikan setiap node dalam linked list dengan memiliki data serta referensi ke simpul sebelumnya dan setelahnya. Di sisi lain, kelas DoublyLinkedList bertanggung jawab untuk mengelola operasi-operasi pada linked list tersebut, seperti menambahkan node di antara dua node yang ada (addInBetween()) dan menghapus node berdasarkan data yang diberikan (deleteNode()). Operasi-operasi ini memanfaatkan konsep referensi maju (next) dan mundur (prev) antara node-node dalam linked list. Setelah kedua kelas didefinisikan, sebuah objek DoublyLinkedList (li) dibuat dan diinisialisasi dengan beberapa simpul awal. Kemudian, operasi penambahan dan penghapusan node dilakukan dengan menggunakan metode yang sesuai, yaitu addInBetween() dan deleteNode(), diikuti oleh pencetakan isi linked list setelah setiap operasi dilakukan.

Tanda Tangan

Setelah di jalankan akan menghasilkan output seperti berikut:

```
Cumi
Kerapu
Lele

=====Add in Between=====
Cumi
Kerapu
Bawal
Lele

=====Delete Node=====
Cumi
Bawal
Lele
```

Gambar 4.2: Output Latihan 1

b. Latihan 2

```
#Latihan 2
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

n1 = Node('Hino')
n2 = Node('Mitsubishi')
n3 = Node('Isuzu')

print(n1.data)
print(n2.data)
print(n3.data)
print()

class CircularLinkedList:
    def __init__(self):
        self.head = None

    def printList(self):
        cur = self.head
        while True:
            print(cur.data)
            cur = cur.next
            if cur == self.head:
                break

    def insertEnd(self, data):
        if not self.head:
            self.head = Node(data)
            self.head.next = self.head
        else:
            newNode = Node(data)
            cur = self.head
            while cur.next != self.head:
                cur = cur.next
            cur.next = newNode
            newNode.next = self.head

    def deleteEnd(self):
        if self.head:
            cur = self.head
            prev = None
            while cur.next != self.head:
                prev = cur
                cur = cur.next
            prev.next = cur.next

li = CircularLinkedList()
li.head = n1
n1.next = n2
n2.next = n3
n3.next = n1

print('=====Add Last=====')
li.insertEnd('Toyota')
li.printList()
print()

print('=====Delete Last=====')
li.deleteEnd()
li.printList()
```

Gambar 4.3: Latihan 2

Kode di atas adalah implementasi dari linked list sirkular dalam bahasa Python. Pertama, kita mendefinisikan kelas Node yang memiliki atribut data dan referensi ke simpul berikutnya. Kemudian, kita membuat tiga objek Node yaitu n1, n2, dan n3 dengan data 'Hino', 'Mitsubishi', dan 'Isuzu' berturut-turut. Selanjutnya, kita mendefinisikan kelas CircularLinkedList yang memiliki metode untuk mencetak isi linked list (printList), menambahkan simpul baru di akhir linked list (insertEnd), dan menghapus simpul terakhir dari linked list (deleteEnd).

Tanda Tangan

Setelahnya, kita membuat objek `CircularLinkedList` baru dengan variabel `li`. Kemudian, kita menghubungkan `n1`, `n2`, dan `n3` secara berurutan untuk membentuk linked list sirkular dengan menyambungkan simpul terakhir (`n3`) ke simpul pertama (`n1`). Kemudian, kita mencetak isi linked list sebelum dan setelah operasi penambahan dan penghapusan simpul terakhir dilakukan.

Setelah di jalankan akan menghasilkan output seperti berikut:

```
Hino
Mitsubishi
Isuzu

=====Add Last=====
Hino
Mitsubishi
Isuzu
Toyota

=====Delete Last=====
Hino
Mitsubishi
Isuzu
```

Gambar 4.4: Output Latihan 2

Tanda Tangan

V. Kesimpulan

Praktikum ini memberikan pemahaman yang baik tentang konsep dasar dan implementasi struktur data linked list. Saya dapat mengambil beberapa kesimpulan dari praktikum ini, antara lain:

- a. Linked list merupakan salah satu struktur data penting dalam pemrograman yang memungkinkan penyimpanan dan manipulasi data secara dinamis.
- b. Dengan memahami konsep dasar linked list, kita dapat mengimplementasikan operasi-operasi dasar seperti penyisipan dan penghapusan node dengan mudah.
- c. Penting untuk memahami algoritma yang mendasari operasi-operasi dalam linked list untuk menghindari kesalahan implementasi yang dapat mengakibatkan kegagalan dalam program.

Dengan demikian, praktikum ini berhasil memberikan wawasan yang baik tentang penggunaan dan implementasi linked list dalam pemrograman.

Tanda Tangan
