

Praktikum Struktur Data

Modul 8 - [Linear Data Structure] Linked List

Senin, 2 Juni 2025

1 Tujuan

1. Mahasiswa *Linear Data Structure* yaitu Linked List, antara lain:
 - (a) Node
 - (b) Linked List
2. Mahasiswa memahami penggunaan dan implementasi dengan menggunakan bahasa pemrograman Python akan Linked List

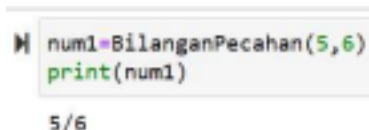
2 Ketentuan

Kerjakan semua soal dalam modul ini baik Tugas Pra-Praktikum maupun Tugas Praktikum, dengan ketentuan sebagai berikut :

1. Semua jawaban ditulis dalam bentuk ipynb format di dalam Google Colab dan print pdf dari file tersebut, dan tiap jawaban diberikan heading yang berisi nomor soal, dan soal dituliskan kembali
2. Setiap nomor dikerjakan dalam dua cell berbeda, cell pertama berisi nomor soal (dalam bentuk heading) dan soal, serta cell kedua adalah jawaban
3. *Submit* link collaboratory di dalam *Google Classroom* dan file pdfnya, sesuai dengan waktu yang telah ditentukan
4. Berikan Nama File berupa dengan format : NPM ModulXX PraPraktikum dan NPM ModulXX Praktikum, misalkan 240411100000 Modul08 PraPraktikum.pdf

3 Tugas Pra-Praktikum

1. Buatlah class Bilangan Pecahan yang merepresentasikan bilangan pecahan (yang terdiri dari bagian pembilang dan bagian penyebut), tambahkan constructor pada class tersebut untuk inialisasi bilangan pecahan
2. Tambahkan method override `str` untuk menampilkan bilangan pecahan yang sudah dibuat, seperti yang terdapat pada Gambar 1



```
num1=BilanganPecahan(5,6)
print(num1)
```

5/6

Gambar 1: Syntax *print* untuk bilangan pecahan

3. Tambahkan method untuk override `add` yang berfungsi menjumlahkan dua buah bilangan pecahan. Sebagai tambahan, untuk menjumlahkan dua bilangan pecahan, terlebih dahulu dihitung Kelipatan

Persekutuan Terkecil dari dua buah penyebut. Contoh hasil eksekusi dapat dilihat pada Gambar ??

```
num1=BilanganPecahan(1,3)
print(num1)
num2=BilanganPecahan(5,8)
print(num2)
num3=num1+num2
print(num3)
```

1/3
5/8
23/24

```
num1=BilanganPecahan(1,3)
print(num1)
num2=BilanganPecahan(5,8)
print(num2)
num3=num1+num2
print(num3)
```

1/3
5/8
23/24

(a) (b)

```
num1=BilanganPecahan(1,4)
print(num1)
num2=BilanganPecahan(1,6)
print(num2)
num3=num1+num2
print(num3)
```

1/4
1/6
5/12

(c)

Gambar 2: Operasi Penjumlahan bilangan pecahan

4 Materi - Linked List

Bahasa pemrograman python telah menyediakan type data yang dinamis, yaitu List. Ukuran dari variabel yang bertipe data list dapat diatur sesuai dengan keinginan programmer selama program dijalankan, tidak harus mempunyai ukuran tetap di awal. Tipe data ini juga menyediakan method menambah data pada saat diperlukan, sehingga tipe data ini bersifat dinamis.

Akan tetapi tidak semua pemrograman menyediakan type data seperti ini, oleh karena itu terdapat suatu struktur data yang dapat dibuat oleh programmer yang bersifat dinamis, yaitu Linked List.

4.1 Node

Linked List adalah struktur data yang terdiri dari beberapa node, dimana Node ini harus memiliki setidaknya dua informasi, yaitu informasi mengenai data atau nilai, dan informasi mengenai node

berikutnya. Oleh karena itu node dibuat menjadi sebuah tipe data baru yang bertipe class, dengan dua informasi yaitu *data* dan *next*. Terdapat beberapa method penting pada class node ini, antara lain:

1. constructor, yang akan dijalankan setiap instansiasi class
2. *getData*, untuk mengetahui informasi data yang terdapat pada node tersebut
3. *getNext*, untuk mengetahui informasi node berikutnya, jika tidak ada node berikutnya maka nilai balik berupa *None*
4. *setData*, untuk merubah informasi data yang terdapat pada node tersebut
5. *setNext*, untuk menentukan node berikutnya yang ditunjukkan oleh informasi *next* dari node tersebut

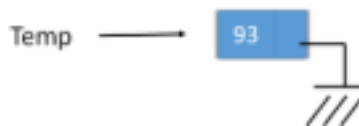
class dari Node ini dapat dilihat pada code berikut :

```
1 class Node :
2 def __init__ ( self , init_data ):
3 self . data = init_data
4 self . next = None
5 def getData ( self ):
6 return self . data
7 def getNext ( self ):
8 return self . next
9 def setData ( self , newdata ) :
10 self . data = newdata
11 def setNext ( self , new_next ):
12 self . next = new_next
```

Misalkan terdapat perintah :

```
temp=Node(93)
```

Maka akan terdapat object temp berupa Node dengan atribut *data* berisi 93, dan *next* tidak menunjuk ke arah manapun, atau menunjuk ke *ground* seperti yang ditunjukkan pada Gambar 3



Gambar 3: Sebuah Node dengan data adalah 93, dan next menuju ke *ground*

4.2 Linked List

Linked list merupakan kumpulan dari node-node yang terhubung satu sama lain. Untuk mengakses node node yang terdapat pada linked list tersebut, haruslah diketahui terlebih dahulu lokasi node pertama dari suatu linked list, sehingga diperlukan pointer tambahan untuk menunjukkan keberadaan node pertama. Head merupakan pointer yang menunjukkan awal dari suatu linked list, yaitu node pertama dari suatu linked list. Pada saat pertama kali linked list ini dibuat, maka head tidak akan menunjukkan node apapun juga, atau menunjuk pada *ground*. Oleh karena itu constructor pada class Linked List adalah inisialisasi atribut head sama dengan *None*, seperti yang ditunjukkan pada Code berikut ini. Pada code tersebut juga ditambahkan method untuk pengecekan apakah linked list terdapat node ataukah tidak, dengan cara mengecek apakah atribut head menunjuk kepada ground ataukah tidak

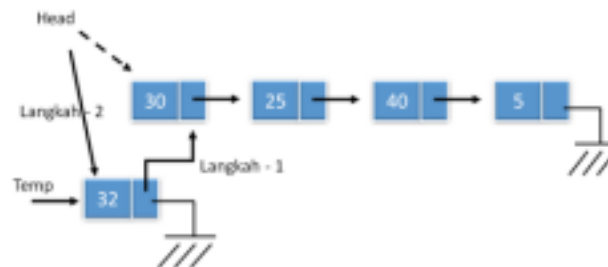
```
1 class Node :
2 class LinkedList :
3 def __init__ ( self ) :
4 self . head = None
5 def isEmpty ( self ):
6 return self . head == None
```

4.2.1 Penambahan Data

Penambahan data dapat dilakukan pada awal linked list, akhir dari linked list, ataupun di posisi antara dua buah node yang sudah ada dalam linked list. Berikut akan ditunjukkan untuk penambahan data atau node baru, dengan node baru berada pada posisi awal dari linked list. Tahapan penambahan data baru pada posisi awal ini adalah :

3

1. Tautkan node baru ini ke node awal dari linked list
2. modifikasi head dari linked list agar menunjuk pada node baru ini, seperti yang ditunjukkan pada Gambar 4 berikut.



Gambar 4: Tahapan penambahan node baru pada posisi awal linked list

Urutan tahapan ini tidak boleh dibalik, karena jika dibalik maka linked list yang awal tidak lagi dapat ditemukan seperti yang terdapat pada Gambar 5



Gambar 5: Tahapan penambahan node baru yang tidak tepat

Method untuk penambahan data pada posisi awal linked List dapat dilihat sebagai berikut :

```
1 class LinkedList :
2 def __init__ ( self ) :
3 self . head = None
4 def isEmpty ( self ) :
5 return self . head == None
6 def addFront ( self , item ) :
7 temp = Node ( item )
8 temp . setNext ( self . head )
9 self . head = temp
```

Misalkan terdapat suatu linked List yang terdiri dari tiga buah node, dengan data mulai dari awal linked list adalah 12, 4, dan 10, maka untuk membuat linked list tersebut dapat dilakukan dengan syntax sebagai berikut :

```
1 dataList = LinkedList ()
2 print ( dataList . head )
3 dataList . addFront ( 10 )
4 dataList . addFront ( 4 )
5 dataList . addFront ( 12 )
6 print ( dataList . head )
```

Jika dieksekusi, maka pada saat print pertama kali, head akan menunjuk pada ground, sedangkan pada print berikutnya menunjuk kepada suatu node.

None

<__main__.Node object at 0x0000022E6AC0B760>

Untuk mengetahui data dari masing-masing node tersebut dapat dilakukan dengan perintah sebagai berikut :

4

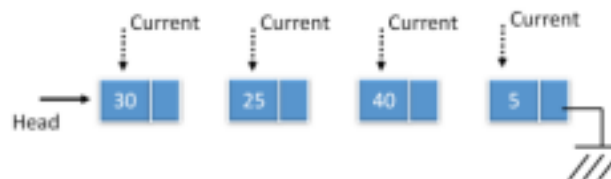
```
1 node1 = dataList . head
2 node2 = node1 . getNext ()
3 node3 = node2 . getNext ()
4 print ( node1 . getData () , node2 . getData () , node3 . getData () )
```

Jika dieksekusi maka akan dapat dilihat urutan node yang mulai dari 12 sampai dengan 10, seperti berikut: 12 4 10

4.2.2 Traversal Linked List

Pada contoh code sebelumnya, untuk mencetak setiap data yang terdapat pada node, dilakukan secara manual dengan menyimpan node dan print setiap node, tentu saja, hal ini kurang efektif, oleh karena itu dibutuhkan proses traversal linked list. Traversal linked list adalah penelusuran setiap node satu persatu, dimulai dari posisi awal sampai dengan posisi akhir. Traversal ini sangat dibutuhkan untuk berbagai keperluan, seperti mencetak data yang terdapat pada setiap node, mengetahui jumlah node yang terdapat pada linked list, penambahan node agar node baru berada pada posisi paling akhir, penyisipan node, dan lain-lain.

Pada proses penelusuran atau traversal dibutuhkan pointer bantuan. Pointer bantuan yang ditunjukkan pada Gambar 6 tersebut, adalah pointer *current* yang bergerak dari node awal sampai dengan node akhir.



Gambar 6: Traversal Linked List

Berikut tambahan method untuk mencetak semua data pada linked list, dengan cara menelusuri node satu persatu mulai dari posisi awal sampai dengan posisi akhir .

```
1 class LinkedList :
2 def __init__ ( self ) :
3 self . head = None
4 def isEmpty ( self ) :
5 return self . head == None
6 def addFront ( self , item ) :
7 temp = Node ( item )
8 temp . setNext ( self . head )
9 self . head = temp
10 def __str__ ( self ) :
11 current = self . head
12 tmp = ""
13 while current != None :
14 tmp += str ( current . getData () ) + " "
15 current = current . getNext ()
16 tmp = "[" + tmp + "]"
17 return tmp
```

Contoh pembuatan linked list dan menampilkan setiap datanya :

```

1 dataList = LinkedList ()
2 dataList . addFront (10)
3 dataList . addFront (4)
4 dataList . addFront (12)
5 dataList . addFront (3)
6 print ( dataList )

```

Maka hasil eksekusinya dapat dilihat sebagai berikut :

```
[ 3 12 4 10 ]
```

5

5 Tugas Praktikum

Buat program dengan menggunakan bahasa pemrograman Python untuk soal-soal berikut ini:

1. Tambahkan method *addLast* untuk menambahkan node baru pada linked list, dimana node tersebut akan terletak pada posisi akhir dari linked list. Petunjuk : tambahkan pointer selain current pada saat penelusuran node, sedemikian hingga posisi pointer baru tersebut akan menunjuk node sebelum node yang ditunjuk oleh current. Contoh hasil eksekusi dapat dilihat pada Gambar 7

```

dataList=LinkedList()
dataList.addFront(10)
dataList.addFront(4)
dataList.addFront(12)
dataList.addFront(3)
dataList.addFront(5)
print(dataList)
dataList.addLast(32)
dataList.addLast(7)
print(dataList)

```

```

[ 5 3 12 4 10 ]
[ 5 3 12 4 10 32 7 ]

```

Gambar 7: Penambahan node baru pada posisi akhir

2. Tambahkan method *search* untuk mencari data pada suatu linked list. Jika data ditemukan maka kembalikan nilai posisi node (dengan node awal berada pada posisi 0), jika data tidak ada, maka kembalikan None, seperti yang ditunjukkan pada Gambar 8.
3. Tambahkan method *insertBefore(node, newNode)* untuk menambahkan node baru *newNode* ke suatu linked list dengan lokasi sebelum *node*, dan tambahkan method *insertAfter(node, newNode)* untuk menambahkan node baru *newNode* ke linked list pada lokasi setelah *node*, seperti yang ditunjukkan pada Gambar 9

Selamat Mengerjakan, Selalu Latihan, Jujur
harus dimulai kapanpun, Bertanya jika kurang
mengerti

6

```
dataList=LinkedList()
dataList.addFront(10)
dataList.addFront(4)
dataList.addFront(12)
dataList.addFront(3)
dataList.addFront(5)
dataList.addLast(32)
dataList.addLast(7)
print(dataList)
print(dataList.search(9))
print(dataList.search(10))
print(dataList.search(5))
```

[5 3 12 4 10 32 7]
None
4
0

Gambar 8: Pencarian data pada linked list

```
dataList=LinkedList()
dataList.addFront(10)
dataList.addFront(4)
dataList.addFront(12)
dataList.addFront(3)
dataList.addFront(5)
dataList.addLast(32)
dataList.addLast(7)
print(dataList)
dataList.insertBefore(10,99)
print(dataList)
dataList.insertAfter(10,100)
print(dataList)

[ 5 3 12 4 10 32 7 ]
[ 5 3 12 4 99 10 32 7 ]
[ 5 3 12 4 99 10 100 32 7 ]
```

Gambar 9: Penyisipan data pada linked list