

Modul Praktikum Struktur Data

Semester Gasal 2022/2023

Yuan Lukito, S.Kom., M.Cs, Kristian Adi Nugraha, S.Kom., M.T., Antonius Rachmat C., S.Kom., M.Cs., I Kadek Dendy

Senaparthi. S. T., M. Eng.



Program Studi
Informatika



UNIVERSITAS KRISTEN
DUTA WACANA

Copyright © 2022 Yuan Lukito, Kristian Adi Nugraha, Antonius Rachmat C., I Kadek Dendy S.

PUBLISHED BY FAKUKTAS TEKNOLOGI INFORMASI UKDW

[HTTP://FTI.UKDW.AC.ID](http://FTI.UKDW.AC.ID)

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the “License”). You may not use this file except in compliance with the License. You may obtain a copy of the License at <http://creativecommons.org/licenses/by-nc/3.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Electronic edition, version: Agustus 2022

2.3.4	Fungsi Rekursif	21
2.4	Guided	22
2.5	Unguided	26
3	Struktur Data Built-in Python	29
3.1	Tujuan Praktikum	29
3.2	Alat dan Bahan	29
3.3	Materi	29
3.3.1	Array pada Python	29
3.3.2	List	30
3.3.3	Array 2 Dimensi	31
3.3.4	Sets	33
3.3.5	Tuple	34
3.3.6	Dictionary	36
3.4	Guided	40
3.5	Unguided	41
4	Penyimpanan Data: File, CSV dan JSON	43
4.1	Tujuan Praktikum	43
4.2	Alat dan Bahan	43
4.3	Materi	43
4.3.1	Membaca Isi File CSV Dengan Modul csv	48
4.4	Guided	51
4.5	Unguided	54
5	Pengantar Class dan Object	55
5.1	Tujuan Praktikum	55
5.2	Alat dan Bahan	55
5.3	Materi	56
5.3.1	Paradigma Pemrograman	56
5.3.2	Class	56
5.3.3	Object	57
5.3.4	Atribut	57
5.3.5	Method	57
5.3.6	Konstruktor	58
5.4	Guided	59
5.5	Unguided	61

II

Part 02: Linear Data Structures

6	Linked List	65
6.1	Tujuan Praktikum	65
6.2	Alat dan Bahan	65

6.3	Materi dan Latihan Guided	66
6.3.1	Linked List	66
6.3.2	Single Linked List	66
6.3.3	Double Linked List	70
6.3.4	Single Linked List dan Double Linked List Circular	74
6.4	Latihan Unguided	74
7	Stack	75
7.1	Tujuan Praktikum	75
7.2	Alat dan Bahan	75
7.3	Materi dan Latihan Guided	75
7.3.1	Stack	75
7.3.2	Stack dengan List	76
7.3.3	Stack dengan Linked List	77
7.4	Latihan Unguided	79
8	Queue dan Deque	81
8.1	Tujuan Praktikum	81
8.2	Alat dan Bahan	81
8.3	Materi dan Latihan Guided	81
8.3.1	Queue	81
8.3.2	Queue dengan List Circular	82
8.3.3	Queue dengan Linked List	83
8.3.4	Deque	85
8.4	Guided	86
8.4.1	Pembahasan	86
8.5	Latihan Unguided	87
9	Priority Queue	89
9.1	Tujuan Praktikum	89
9.2	Alat dan Bahan	89
9.3	Materi	89
9.3.1	Priority Queue	89
9.3.2	Priority Queue Dengan Unsorted Linked List	90
9.3.3	Priority Queue Dengan Sorted Linked List	91
9.4	Guided	93
9.4.1	Implementasi Priority Queue (unsorted)	93
9.4.2	Implementasi Priority Queue (sorted)	97
9.5	Unguided	100

III

Part 03: Sorting and Searching

10	Quick Sort dan Merge Sort	103
10.1	Tujuan Praktikum	103

10.2	Alat dan Bahan	103
10.3	Materi	104
10.3.1	Algoritma Divide and Conquer	104
10.3.2	Quick Sort	106
10.3.3	Merge Sort	107
10.4	Unguided	109
11	Hashing	111
11.1	Tujuan Praktikum	111
11.2	Alat dan Bahan	111
11.3	Materi dan Latihan Guided	111
11.3.1	Hashing	111
11.3.2	Penanganan collision pada hash table	113
11.3.3	Latihan Unguided	115

IV

Part 04: Non Linear Data Structures

12	Tree	119
12.1	Tujuan Praktikum	119
12.2	Alat dan Bahan	119
12.3	Materi	119
12.3.1	Tree	119
12.4	Guided	120
12.5	Unguided	123
13	Binary Tree	125
13.1	Tujuan Praktikum	125
13.2	Alat dan Bahan	125
13.3	Materi	125
13.3.1	Binary Tree	125
13.4	Guided	127
13.5	Unguided	131
14	Graph	133
14.1	Tujuan Praktikum	133
14.2	Alat dan Bahan	133
14.3	Graph	133
14.4	Implementasi Graph	135
14.5	Guided	135
14.6	Unguided	138

15	Revision History	139
15.1	Daftar Revisi Modul Praktikum	139
15.1.1	Semester Gasal 2022/2023	139
15.1.2	Semester Gasal 2021/2022	139
15.1.3	Semester Gasal 2020/2021	139
	Bibliografi	141

Part 01: Python

1	Review Python	11
1.1	Tujuan Praktikum	
1.2	Alat dan Bahan	
1.3	Materi	
1.4	Guided	
1.5	Unguided	
2	Analisis Algoritma dan Fungsi Rekursif	19
2.1	Tujuan Praktikum	
2.2	Alat dan Bahan	
2.3	Materi	
2.4	Guided	
2.5	Unguided	
3	Struktur Data Built-in Python	29
3.1	Tujuan Praktikum	
3.2	Alat dan Bahan	
3.3	Materi	
3.4	Guided	
3.5	Unguided	
4	Penyimpanan Data: File, CSV dan JSON	43
4.1	Tujuan Praktikum	
4.2	Alat dan Bahan	
4.3	Materi	
4.4	Guided	
4.5	Unguided	
5	Pengantar Class dan Object	55
5.1	Tujuan Praktikum	
5.2	Alat dan Bahan	
5.3	Materi	
5.4	Guided	
5.5	Unguided	

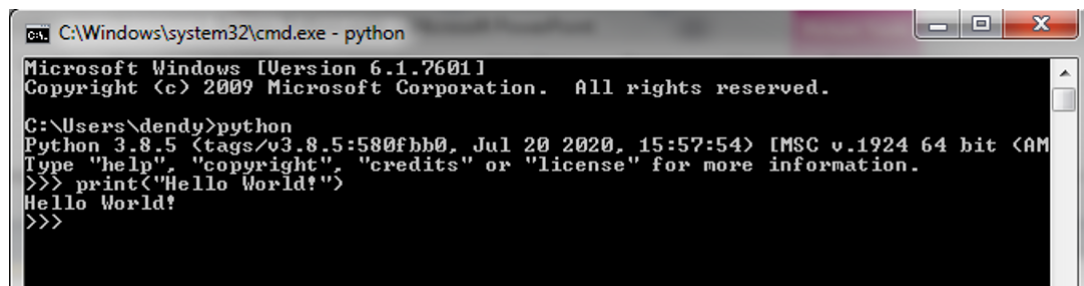
sifatnya yang diinterpretasikan, menjadikannya bahasa yang ideal untuk skrip dan pengembangan aplikasi yang cepat di banyak area di sebagian besar platform.

Bahasa ini muncul pertama kali pada tahun 1991, dirancang oleh seorang bernama Guido van Rossum. Sampai saat ini Python masih dikembangkan oleh Python Software Foundation. Interpreter Python dan pustaka standar yang luas tersedia secara bebas dalam bentuk kode sumber atau biner untuk semua platform utama dari situs Web Python, <https://www.python.org/>, dan dapat didistribusikan secara bebas. Situs yang sama juga berisi distribusi dan referensi ke banyak modul Python gratis dari pihak ketiga, program dan alat, serta dokumentasi tambahan.

Bab ini akan memperkenalkan anda secara singkat ke konsep dan fitur dasar bahasa dan sistem Python. Sehingga akan membantu anda untuk menggunakan interpreter Python yang praktis serta mendapatkan pengalaman secara langsung dengan mencoba semua contoh mandiri.

1.3.2 Interpreter Python

Program Python dieksekusi dengan menggunakan interpreter Python, yang merupakan sebuah program yang khusus membaca dan mengeksekusi kode instruksi yang ditulis dengan bahasa Python. Interpreter ini dapat mengeksekusi program Python dengan menggunakan mode interaktif atau mode script. Mode interaktif sangat membantu programmer saat melakukan testing perbaris pada segmen program. Untuk dapat menggunakan mode interaktif, bukalah aplikasi command prompt pada Windows lalu tuliskan perintah : Python. Ilustrasi stack dapat dilihat pada gambar 13.2



Gambar 1.1: Ilustrasi Mode Interaktif Python

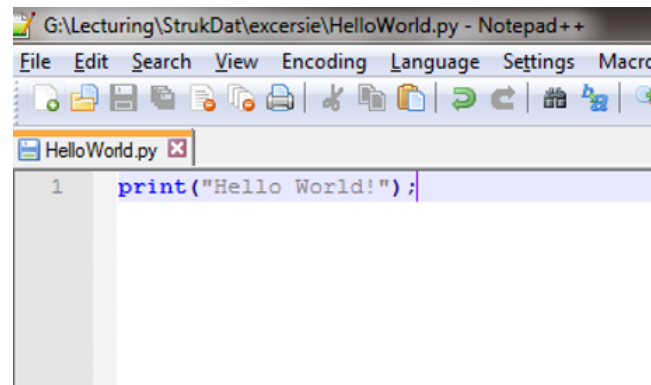
Pada simbol >>> dibagian bawah output, menunjukan mode interaktif untuk memasukan perintah Python ke interpreter. Pada contoh gambar 13.2, Anda memasukan statement

```
1 print ("Hello World!")
```

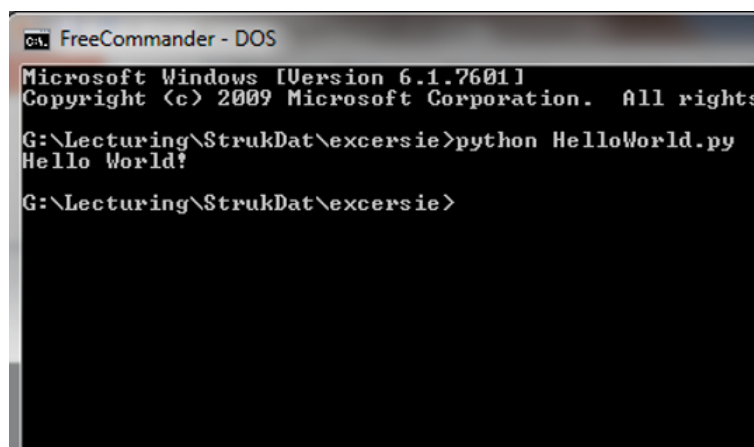
Mode berikutnya adalah mode script, dimana Anda menulis kode program anda pada sebuah file yang ber-ekstensi py. File script (source code) Python ini dapat dibuat dengan menggunakan text editor atau Integrated Development Environment (IDE) apapun. Contoh bagaimana file dibuat dapat dilihat pada gambar 1.2.

Untuk mengeksekusi script tersebut, dapat dilakukan melalui command prompt dan mengetikan perintah sebagai berikut :

```
1 python HelloWorld.py
```



Gambar 1.2: Membuat Script bernama HelloWorld.py menggunakan Notepad++



Gambar 1.3: Mengeksekusi script Python

1.3.3 Tipe Data Primitive

Tipe data adalah suatu media atau memori pada komputer yang digunakan untuk menampung informasi. Python sendiri mempunyai tipe data yang cukup unik bila kita bandingkan dengan bahasa pemrograman yang lain. Berikut adalah tipe data dari bahasa pemrograman Python :

1. Integer : menyimpan data bilangan bulat
2. Float : menyimpan data bilangan pecahan
3. String : menyimpan data berupa karakter/kalimat, bisa berupa huruf angka, dll.
4. List : menyimpan data deret array sekuensial
5. Tuple: menyimpan data mirip seperti List, namun isinya tak dapat diubah
6. Set : menyimpan data tidak terurut dan terindeks
7. Dictionary : menyimpan data terurut, dapat diubah, dan terindeks

1.3.4 Variable

Variabel adalah lokasi memori yang dicadangkan untuk menyimpan nilai-nilai. Ini berarti bahwa ketika Anda membuat sebuah variabel Anda memesan beberapa ruang di memori. Variabel menyimpan data yang dilakukan selama program dieksekusi, yang nantinya isi dari variabel tersebut dapat diubah oleh operasi - operasi tertentu pada program yang menggunakan variabel.

Variabel dapat menyimpan berbagai macam tipe data. Di dalam pemrograman Python, variabel mempunyai sifat yang dinamis, artinya variabel Python tidak perlu dideklarasikan tipe data tertentu dan variabel Python dapat diubah saat program dijalankan.

Penulisan variabel Python sendiri juga memiliki aturan tertentu, yaitu :

1. Karakter pertama harus berupa huruf atau garis bawah atau underscore
2. Karakter selanjutnya dapat berupa huruf, garis bawah/underscore atau angka
3. Karakter pada nama variabel bersifat sensitif (case-sensitif). Artinya huruf kecil dan huruf besar dibedakan

Untuk mulai membuat variabel di Python caranya sangat mudah, Anda cukup menuliskan variabel lalu mengisinya dengan suatu nilai dengan cara menambahkan tanda sama dengan = diikuti dengan nilai yang ingin dimasukan.

1.3.5 Operator

Operator adalah konstruksi yang dapat memanipulasi nilai dari operan. Sebagai contoh operasi $3 + 2 = 5$. Disini 3 dan 2 adalah operan dan + adalah operator. Bahasa pemrograman Python mendukung berbagai macam operator, diantaranya :

Operator	Keterangan	Contoh
+	Penambahan	$2 + 2 = 4$
-	Pengurangan	$2 - 2 = 0$
*	Perkalian	0.296
/	Pembagian real	$2 / 2 = 1$
//	Pembagian tanpa nilai pecahan (floor division)	$5 // 2 = 2$
%	Modulo	$5 \% 2 = 1$
**	Perkalian pangkat (power of)	$5 ** 2 = 25$

Tabel 1.1: Operator aritmatika pada Python

Operator	Keterangan	Contoh
>	Apakah A lebih besar dari B	$A > B$
<	Apakah A lebih kecil dari B	$A < B$
==	Apakah A sama dengan B	$A == B$
>=	Apakah A lebih besar atau sama dengan B	$A >= B$
<=	Apakah A lebih kecil atau sama dengan B	$A <= B$
!=	Apakah A tidak sama dengan B	$A != B$

Tabel 1.2: Operator Relasional pada Python

Operator	Keterangan	Contoh
Not	Operator yg melakukan inversi nilai operand	$(\text{not True}) = \text{False}$
And	Operator untuk gerebang logika and.	$(1 \text{ and } 0) = 0$
Or	Operator gerebang logika or.	$(1 \text{ or } 0) = 1$

Tabel 1.3: Operator Boolean pada Python

1.3.6 Function

Fungsi adalah blok kode terorganisir dan dapat digunakan kembali yang digunakan untuk melakukan sebuah tindakan/action. Fungsi memberikan modularitas yang lebih baik untuk aplikasi Anda dan tingkat penggunaan kode yang tinggi.

Operator	Keterangan	Contoh
Is	Membandingkan apakah referensi dari dua variable adalah sama.	Nama1 = "Anton" Nama2 = "Yuan" Hasil = nama1 is nama 2
Is not	Membandingkan referensi dari dua variable tidak sama.	Nama1 = "Dendy" Nama2 = "Yuan" Hasil = nama1 is not nama 2

Tabel 1.4: Operator Logika pada Python

Anda dapat menentukan fungsi untuk menyediakan fungsionalitas yang dibutuhkan. Berikut adalah aturan sederhana untuk mendefinisikan fungsi dengan Python.

1. Fungsi blok dimulai dengan def kata kunci diikuti oleh nama fungsi dan tanda kurung ().
2. Setiap parameter masukan atau argumen harus ditempatkan di dalam tanda kurung ini. Anda juga dapat menentukan parameter di dalam tanda kurung ini.
3. Pernyataan pertama dari sebuah fungsi dapat berupa pernyataan opsional - string dokumentasi fungsi atau docstring.
4. Blok kode dalam setiap fungsi dimulai dengan titik dua (:) dan indentasi.
5. Pernyataan kembali [ekspresi] keluar dari sebuah fungsi, secara opsional menyampaikan kembali ekspresi ke pemanggil. Pernyataan pengembalian tanpa argumen sama dengan return None.

```

1 def printme( str ):
2     print (str)
3     return

```

1.3.7 Modul & Python Standard Library

Modul memungkinkan Anda mengatur kode Python secara logis. Mengelompokkan kode terkait ke dalam modul membuat kode lebih mudah dipahami dan digunakan. Modul adalah objek Python dengan atribut yang diberi nama yang bisa Anda bind dan dijadikan referensi.

Secara sederhana modul adalah file yang terdiri dari kode Python. Modul dapat mendefinisikan fungsi, kelas dan variabel. Modul juga bisa menyertakan kode yang bisa dijalankan "runable".

Berikut adalah contoh modul sederhana pada Python :

```

1 #file ini disimpan dengannama support.py
2 def print_func( par ):
3     print (Halo, selamat datang di : ", par)
4     return

```

Anda dapat menggunakan file sumber Python apapun sebagai modul dengan mengeksekusi pernyataan impor di file sumber Python lainnya. Impornya memiliki sintaks berikut.

```

1 import support
2
3 support.print_func("Struktur Data")

```

Python menyediakan fungsionalitas tambahan yang dapat dengan mudah digunakan dan ditambahkan pada program. Python Standard Library adalah pustaka tambahan yang menyediakan berbagai komponen yang siap pakai. Kata kunci `import` digunakan untuk menambahkan komponen yang ada pada Python Standard Library.

```
1 from math import *
2 from math import sqrt
3 from math
```

1.3.8 Input & Output

Python menyediakan banyak fungsi built-in yang bisa kita gunakan. Salah satunya adalah yang berkenaan dengan fungsi i/o atau input output. Fungsi bawaan untuk melakukan operasi output adalah `print()`, dan fungsi untuk melakukan operasi input adalah fungsi `input()`. Kita akan membahas fungsi `print()` terlebih dahulu.

```
1 a = input("Kalimat ini dicetak ke layar")
2 print("Nilai a =", a)
```

1.3.9 Struktur Kontrol

Struktur kontrol adalah cara yang digunakan untuk mengambil keputusan apabila di dalam program dihadapkan pada kondisi tertentu. Jumlah kondisinya bisa satu, dua atau lebih.

Percabangan mengevaluasi kondisi atau ekspresi yang hasilnya benar atau salah. Kondisi atau ekspresi tersebut disebut ekspresi boolean. Hasil dari pengecekan kondisi adalah `True` atau `False`. Bila benar (`True`), maka pernyataan yang ada di dalam blok kondisi tersebut akan dieksekusi. Bila salah (`False`), maka blok pernyataan lain yang dieksekusi.

Secara umum, Python mengeksekusi program baris perbaris. Mulai dari baris satu, dua, dan seterusnya. Ada kalanya, kita perlu mengeksekusi satu baris atau satu blok kode program beberapa kali. Hal ini disebut dengan perulangan atau biasa disebut looping atau iterasi.

Operator	Contoh
If-else	<pre>if -6 < 0 : print("Nilai adalah negatif.") else: print("Nilai adalah positif.") count = 10</pre>
while	<pre>while count >= 0 : print(count) count -= 1</pre>
for	<pre>for i in range(0, 10) : print(i)</pre>

Tabel 1.5: Struktur kontrol pada Python

1.4 Guided

Pada bagian ini akan diberikan beberapa contoh program yang harus Anda coba untuk mendapatkan pemahaman lebih baik tentang materi pada bab ini.

Tipe Data

Cobalah kode program berikut, jalankan, dan perhatikan outputnya!

```
1 integer = 123123123123123123 #Integer
2 float = 1123.123 #float
3 string = "Hello World" #string
4 boolean = True #boolean
5 thislist = ["apple", "banana", "cherry"] #list
6 thistuple = ("apple", "banana", "cherry") #tuple
7 thisset = {"apple", "banana", "cherry"} #set
8 thisdict = {
9     brand: "Ford",
10    model: "Mustang",
11    year: 1964
12 } #dictionary
13
14 print("integer ",type(integer))
15 print("float ",type(float))
16 print("string ",type(string))
17 print("boolean ",type(boolean))
18 print("list ",type(thislist))
19 print("tuple ",type(thistuple))
20 print("set ",type(thisset))
21 print("dictionary ",type(thisdict))
```

Variable

Cobalah kode program berikut, jalankan, dan perhatikan outputnya!

```
1 nama = "John Doe"
2 print(nama)
3
4 umur = 20
5 print(umur)
6 type(umur)
7 umur = "dua puluh satu"
8 print(umur)
9 type(umur)
10
11 namaDepan = "Dendy"
12 namaBelakang = "Senapatha"
13 nama = namaDepan + " " + namaBelakang
14 umur = 22
15 hobi = "Berenang"
16 print("Biodata\n", nama, "\n", umur, "\n", hobi)
17
18 inivariabel = "Halo"
19 ini_juga_variabel = "Hai"
20 inivariabeljuga = "Hi"
21 inivariabel222 = "Bye"
```

```
22
23 panjang = 10
24 lebar = 5
25 luas = panjang * lebar
26 print(luas)
```

Function

Cobalah kode program berikut, jalankan, dan perhatikan outputnya!

```
1 def isPalindrome(s):
2     return s == s[::-1]
3
4 # Driver code
5 s = input("Masukan kata yang akan di check!");
6 ans = isPalindrome(s)
7
8 if ans:
9     print("Ya")
10 else:
11     print("Tidak")
```

1.5 Unguided

Sebagai latihan mandiri silahkan menjawab soal-soal unguided berikut ini dengan menggunakan contoh dan teori yang sudah dijelaskan sebelumnya.

- Exercise 1.1** Buatlah aplikasi kalkulator sederhana yang memiliki fitur-fitur sebagai berikut :
1. Fungsi untuk melakukan penambahan, pengurangan, perkalian dan pembagian.
 2. Aplikasi akan terus dijalankan dan berhenti apabila user menekan tombol "Q"

algoritma A dikatakan lebih baik dari algoritma B jika waktu yang diperlukan untuk menjalankan algoritma A lebih sedikit dibandingkan dengan algoritma B. Sedikit atau banyak waktu yang diperlukan sebagian besar ditentukan oleh kompleksitas langkah, jumlah input yang harus diolah dan jumlah langkah yang harus dilakukan.

Analisis algoritma bertujuan untuk melakukan analisis efisiensi terhadap suatu algoritma. Analisis dapat dilakukan dengan dua cara, yaitu:

1. Menghitung langkah-langkah yang diperlukan oleh suatu algoritma berdasarkan source code dari algoritma tersebut.
2. Melakukan percobaan secara empiris dengan mengukur waktu yang diperlukan berdasarkan ukuran input yang diberikan.

2.3.2 Notasi Asymptotic

Notasi asymptotic digunakan untuk menyatakan tingkat efisiensi suatu algoritma berdasarkan waktu yang diperlukan untuk menjalankannya. Notasi asymptotic menyatakan hubungan antara jumlah input, kompleksitas langkah dan waktu. Secara umum ada tiga macam notasi asymptotic yang dapat digunakan:

1. Big-Oh notation (batas atas, berdasarkan worst-case).
2. Omega notation (batas bawah, berdasarkan best-case).
3. Theta notation (di antara batas atas dan batas bawah, berdasarkan average-case).

2.3.3 Analisis Secara Empiris

Analisis empiris dilakukan dengan cara melakukan pengukuran waktu dari suatu algoritma, dengan jumlah input yang bertambah besar. Dari laju pertambahan waktu yang diperlukan jika input ditambah, dapat diketahui tingkat efisiensi dan performa dari algoritma tersebut. Untuk melakukan pengukuran waktu, Python menyediakan beberapa pilihan. Untuk modul ini akan dibahas dua alternatif berikut ini:

1. Menggunakan fungsi `time()` dari modul `time`.
2. Menggunakan fungsi `default_timer()` dari modul `timeit`.

Contoh Penggunaan `time()`

Untuk dapat menggunakan fungsi `time()`, yang harus dilakukan terlebih dahulu adalah meng-import module `time` ke dalam program kita. Fungsi `time()` akan menghasilkan waktu saat fungsi tersebut dipanggil. Untuk lebih jelasnya, perhatikan contoh penggunaan fungsi `time()` pada source code berikut ini:

```
1  import time
2
3  start = time.time() # catat waktu mulainya
4  nilai = 0
5  for i in range(100000):
6  nilai = nilai + i * 2
7  end = time.time() # catat waktu selesainya (dalam detik)
8
9  # hitung selisih waktunya
10 print(end - start)
11
```

Contoh Penggunaan `timeit()`

Untuk dapat menggunakan `timeit`, yang harus dilakukan terlebih dahulu adalah meng-import module `timeit` ke dalam program kita. Kemudian pengukuran waktu dilakukan dengan cara

memanggil fungsi `default_timer()` di awal dan di akhir bagian kode program yang akan diukur waktunya. Untuk lebih jelasnya perhatikan source code berikut ini:

```

1     import timeit
2
3     start = timeit.default_timer() # catat waktu mulainya
4     nilai = 0
5     for i in range(100000):
6         nilai = nilai + i * 2
7     end = timeit.default_timer() # catat waktu selesainya
8
9     # tampilkan selisih waktunya (dalam detik)
10    print(end - start)
11

```

2.3.4 Fungsi Rekursif

Secara umum fungsi rekursif adalah fungsi yang memanggil dirinya sendiri. Fungsi rekursif akan berjalan terus-menerus sampai pada suatu titik berhenti, dimana fungsi tersebut berhenti memanggil dirinya sendiri. Sebuah fungsi rekursif terdiri dari dua bagian, yaitu base case dan recursive case.

Contoh masalah yang sering dijadikan contoh fungsi rekursif adalah masalah faktorial. Pada matematika dikenal operasi faktorial sebagai berikut:

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

$$n! = n \times (n-1) \times (n-2) \times \dots \times 1$$

Sehingga dapat disederhanakan menjadi $n! = n \times (n-1)!$

Masalah faktorial dapat dipecahkan dengan dua cara, yaitu cara iteratif biasa dan cara rekursif. Perhatikan perbedaan antara fungsi biasa dan fungsi rekursif seperti berikut ini:

```

1     def faktorial_iteratif(n): # fungsi iteratif (non-rekursif)
2         hasil = 1
3         for i in range(n, 0, -1):
4             hasil = hasil * i
5         return hasil
6
7     def faktorial_rekursif(n): # fungsi rekursif
8         if n == 1:
9             return 1
10        else:
11            return n * faktorial_rekursif(n-1)
12
13
14    hasil1 = faktorial_iteratif(10)
15    print('Faktorial 10 dengan cara iteratif: ', hasil1)
16    hasil2 = faktorial_rekursif(10)
17    print('Faktorial 10 dengan cara rekursif: ', hasil2)
18

```

Base case adalah bagian dari fungsi rekursif yang langsung memberikan hasil tanpa memanggil lagi secara rekursif. Pada contoh fungsi `faktorial_rekursif()`, dapat dilihat bahwa base case pada saat

n bernilai 1. Artinya 1! sudah pasti hasilnya adalah 1, tidak perlu menghitung lagi. Berbeda dengan 5!, maka harus menghitung dulu dengan $5! = 5 \times 4!$. Perhitungan 4! dilakukan dengan memanggil fungsi faktorial_rekursif(4). Bagian ini dinamakan sebagai recursive case. Bagian ini biasanya dapat dilihat dari pemanggilan secara rekursif dengan nilai parameter yang semakin mendekati nilai dari kondisi base-casenya.

Head recursion vs Tail recursion

Berdasarkan letak dan urutan dari pemanggilan rekursif, maka fungsi rekursif dapat dibedakan menjadi dua macam, yaitu head recursion dan tail recursion. Perbedaan keduanya dapat dilihat dari source code berikut ini:

```
1      # head recursion
2      def head_recursion(n):
3          if n == 0:
4              return
5          else:
6              head_recursion(n-1)
7              print(n)
8
9
10     def tail_recursion(n):
11         if n == 0:
12             return
13         else:
14             print(n)
15             tail_recursion(n-1)
```

Pada head-recursion, pemanggilan fungsi secara rekursif dilakukan di awal terlebih dahulu. Sedangkan pada tail-recursion, pemanggilan fungsi secara rekursif dilakukan di bagian akhir. Bentuk tail-recursion lebih umum dijumpai dan digunakan dibanding bentuk head-recursion.

Limit fungsi rekursif di Python

Pemanggilan fungsi rekursif pada Python dibatasi sampai 1000 tingkat saja, dengan pertimbangan sudah mencukupi kebutuhan pada umumnya. Jika fungsi rekursif terlalu dalam juga akan menyebabkan jumlah memory yang dibutuhkan semakin banyak dan waktu eksekusi juga akan meningkat. Seandainya diperlukan lebih dari 1000 tingkat, bisa dilakukan dengan menggunakan fungsi setrecursionlimit() dari modul sys, seperti contoh berikut ini:

```
1      import sys
2      x=1500
3      sys.setrecursionlimit(x)
4
5      # limit rekursif sekarang sudah menjadi 1500 tingkat.
```

2.4 Guided

Pada bagian ini akan diberikan beberapa latihan soal dan sekaligus pembahasan soal-soal tersebut.

Pencarian pasangan di dalam List

Anda diberi sebuah list berisi integer berukuran besar yang isinya sudah diurutkan secara ascending (dari kecil ke besar). Jika diberi suatu nilai target, anda harus mencari di dalam list tersebut, apakah ada sepasang nilai (n_1 dan n_2) yang memenuhi $n_1 + n_2 = \text{target}$.

Sebagai contoh, misalnya adalah list berisi [3, 6, 12, 19, 20, 22, 29, 33, 35]. Jika anda diberi target = 25, maka di dalam list ada angka $n_1 = 3$ dan $n_2 = 22$ yang memenuhi $n_1 + n_2 = 25$. Jika target = 42, maka tidak ada sepasang bilangan yang memenuhi target tersebut. Buatlah fungsi untuk memecahkan masalah ini dan ukurlah kinerjanya dengan menggunakan analisis asymptotic dan analisis empiris! Perhatikan: fungsi anda hanya perlu return True/False.

Pembahasan pencarian pasangan

List yang berukuran besar dapat di-generate secara random dengan cara sebagai berikut:

```

1      # butuh modul random
2      import random
3
4      # generate list terurut berukuran n
5      def generate_list(n):
6          batas_atas = n * 100
7          randomlist = random.sample(range(0, batas_atas), n)
8          randomlist.sort()
9          return randomlist
10
11     data = generate_list(100)
12     print(data)

```

Masalah pencarian pasangan ini sebenarnya merupakan masalah pencarian di dalam list. Pendekatan pertama adalah dengan cara linear search, pencarian dilakukan dari awal list sampai akhir. Misal ada list yang isinya [3, 6, 12, 19, 20, 22, 29, 33, 35]. Apakah ada sepasang bilangan yang jika dijumlahkan hasilnya adalah 41?

Untuk setiap isi dari list, carilah apakah ada pasangan yang jika dijumlahkan dengannya akan sesuai nilainya dengan target? Berikut adalah urutan langkah pencariannya:

1. Pada list tersebut index ke-0 berisi nilai 3. Apakah ada pasangan dari 3 yang dapat memenuhi target? Jika targetnya adalah 41, maka bilangan yang dicari adalah 38. Apakah ada angka 38 di dalam list? Jika dicari dari index ke-1 sampai index terakhir tidak ada angka 38.
2. Lanjutkan ke index ke-1 yang berisi nilai 6. Untuk mencapai 41, maka yang dicari adalah 35. Apakah ada angka 35 di dalam list? Ternyata ada. Maka pencarian selesai dan fungsi mengembalikan nilai True.

Jika ternyata setelah dicek semuanya tidak ada yang memenuhi, fungsi tersebut harus mengembalikan nilai False. Source code program sesuai dengan algoritma tersebut adalah sebagai berikut:

```

1      # butuh modul random
2      import random
3
4      # generate list terurut berukuran n
5      def generate_list(n):
6          batas_atas = n * 100

```



```

7     randomlist = random.sample(range(0, batas_atas), n)
8     randomlist.sort()
9     return randomlist
10
11    def cari_pasangan(data, target):
12    for i in range(0, len(data)-1):
13    for j in range(i, len(data)):
14    if data[i] + data[j] == target:
15    print(data[i], '+', data[j], '=', target)
16    return True
17    return False
18
19    data = generate_list(100)
20    print(data)
21
22    # cari apakah ada pasangan yang memenuhi target = 154
23    hasil = cari_pasangan(data, 154)
24    print(hasil)
25

```

Lalu bagaimana dengan tingkat kinerja algoritma tersebut? Jawabannya bisa diketahui dengan mudah berdasarkan potongan dari program di bagian berikut ini:

```

1    def cari_pasangan(data, target):
2    for i in range(0, len(data)-1):
3    for j in range(i, len(data)):
4    if data[i] + data[j] == target:
5    print(data[i], '+', data[j], '=', target)
6    return True
7    return False

```

Di dalam fungsi cari_pasangan() terdapat for di dalam for, secara umum kinerja dari algoritma ini adalah $O(n^2)$. Bagaimana dengan analisis empiris? Untuk percobaan ini dilakukan pengambilan waktu untuk list berukuran 5000, 10000, ... sampai dengan 50000 yang akan dicatat dalam bentuk tabel. Untuk pencatatan waktu maka program perlu diubah dan dilakukan berulang-ulang, seperti berikut ini:

```

1    # butuh modul random
2    import random
3    import timeit
4
5    # generate list terurut berukuran n
6    def generate_list(n):
7    batas_atas = n * 100
8    randomlist = random.sample(range(0, batas_atas), n)
9    randomlist.sort()
10   return randomlist
11

```



```

12     def cari_pasangan(data, target):
13     for i in range(0, len(data)-1):
14     for j in range(i, len(data)):
15     if data[i] + data[j] == target:
16     print(data[i], '+', data[j], '=', target)
17     return True
18     return False
19
20     for i in range(5000, 55000, 5000):
21     data = generate_list(i)
22
23     #target dibuat supaya mencapai kondisi worst case
24
25     target = -100
26
27     # catat waktu mulai
28     start = timeit.default_timer()
29
30     # cari apakah ada pasangan yang memenuhi target
31     hasil = cari_pasangan(data, target)
32
33     # catat waktu hasil didapat
34     end = timeit.default_timer()
35
36     # tampilkan di layar
37     print('Ukuran ', i, ': ', end-start, ' second.')
38

```

Jika program tersebut dijalankan pada notebook (Core i5, RAM 12 GB) hasil yang didapatkan dapat dilihat pada Gambar 2.1.

```

Ukuran 5000 : 1.1411594 second.
Ukuran 10000 : 4.394299200000001 second.
Ukuran 15000 : 9.974223299999998 second.
Ukuran 20000 : 17.380307900000002 second.
Ukuran 25000 : 28.5617206 second.
Ukuran 30000 : 40.0075816 second.
Ukuran 35000 : 54.97108859999999 second.
Ukuran 40000 : 73.11930719999998 second.
Ukuran 45000 : 88.2021785 second.
Ukuran 50000 : 119.38144480000001 second.

```

Gambar 2.1: Hasil analisis empiris program pencari pasangan bilangan (linear search)

Untuk mendapatkan tingkat kinerja, dilakukan perbandingan ukuran list terhadap waktu yang diperlukan, disusun seperti yang ditunjukkan pada Gambar 2.2. Dapat disimpulkan algoritma tersebut merupakan algoritma dengan tingkat kinerja $O(n^2)$ dengan beberapa bukti: jika ukuran dinaikkan dua kali lipat, waktu yang diperlukan menjadi 4 kali lipat semula. Demikian juga jika ukuran dinaikkan 5 kali lipat, maka waktu yang diperlukan adalah 25 kali lipat semula.

Algoritma $O(n^2)$ masih terhitung kurang bagus untuk memecahkan masalah pencarian pasangan ini, terutama jika ukuran list semakin membesar. Dapatkah anda mencari alternatif solusi dengan

Size (n)	Time (second)	Rasio Ukuran	Rasio Waktu
5000	1.1411594	1	1
10000	4.3942992	2	3.850732159
15000	9.9742233	3	8.740429514
20000	17.3803079	4	15.23039454
25000	28.5617206	5	25.02868626
30000	40.0075816	6	35.0587145
35000	54.9710886	7	48.17126214
40000	73.1193072	8	64.07457819
45000	88.2021785	9	77.29172498
50000	119.3814448	10	104.6141712

Gambar 2.2: Hasil analisis perbandingan ukuran list dengan waktu

kinerja lebih baik?

Petunjuk: coba gunakan binary search untuk meningkatkan kinerja menjadi $O(n \log(n))$. Jika anda kreatif, sebenarnya ada solusi $O(n)$ untuk masalah ini yang hasil pengukuran empirisnya dapat dilihat pada Gambar 2.3 yang hasilnya berbeda jauh dengan kinerja algoritma yang telah dibahas.

```

Ukuran 5000 : 0.0005922999999999901 second.
Ukuran 10000 : 0.0019427999999999945 second.
Ukuran 15000 : 0.0018024999999999987 second.
Ukuran 20000 : 0.0034025000000000027 second.
Ukuran 25000 : 0.0029990000000000017 second.
Ukuran 30000 : 0.0047001000000000004 second.
Ukuran 35000 : 0.0041905000000000041 second.
Ukuran 40000 : 0.0047970999999999999 second.
Ukuran 45000 : 0.0068141999999999925 second.
Ukuran 50000 : 0.0070155000000000063 second.

```

Gambar 2.3: Hasil analisis empiris solusi pencarian pasangan dengan algoritma $O(n)$

2.5 Unguided

Exercise 2.1 Lakukan perbandingan kinerja dari fungsi iteratif dengan fungsi rekursif untuk mendapatkan suku ke- n dari suatu deret fibonacci. Perbandingan dilakukan dengan mencatat waktu yang diperlukan untuk mendapatkan suku fibonacci ke-1 sampai ke-100 (nilai n) dengan kedua fungsi tersebut. Kemudian berdasarkan catatan waktu tersebut, buatlah grafik waktu terhadap n . Terakhir, dari grafik tersebut tentukan Big-O dari masing-masing fungsi.

Anda harus menghasilkan:

1. Source code fungsi fibonacci ke- n secara iteratif dan secara rekursif.
2. Tabel catatan waktu dari kedua fungsi tersebut, dengan nilai $n = 1$ sampai 100
3. Grafik garis dari waktu (sumbu y) terhadap n (sumbu x) berdasarkan tabel catatan waktu yang didapatkan.

Baru kemudian anda bisa menentukan Big-O dari masing-masing fungsi.

Fibonacci ke- n maksudnya adalah suku ke- n dari deret bilangan Fibonacci yang dimulai dari 1, 1, 2, 3, 5, ... dst. Jadi misalnya ditanya berapa suku fibonacci ke-10? Jawabannya adalah:

55, karena 10 suku fibonacci pertama adalah 1, 1, 2, 3, 5, 8, 13, 21, 34, 55. ■

Python tidak menyediakan built-in Array, namun menyediakan Lists sebagai penggantinya. Namun python telah menyediakan module array, yang merupakan bagian dari Python Standard Library, mengimplementasikan fitur-fitur dasar dari array.

Untuk dapat menggunakan array dari standard library, Anda harus meng-importnya terlebih dahulu. Pemberian nilai array cukup dengan menuliskan beberapa nilai yang dipisahkan dengan tanda koma dan semua nilai tersebut dituliskan dengan tanda kurung siku dan semua nilai tersebut dituliskan diantara tanda kurung siku.

```

1 from array import *
2
3 array_num = array('i', [1,3,5,7,9])
4 array_num.typecode

```

Module array menyediakan beberapa method penting yang dapat Anda gunakan untuk berbagai pemrosesan array. Berikut adalah daftar methodnya:

Nama Method	Keterangan
insert(i, x)	Menambah item <i>x</i> pada index <i>i</i> array
pop(i)	Menghapus elemen dengan index <i>i</i> dari array dan membaca nilainya
remove(x)	Menghapus element <i>x</i> dari array
reverse()	Membalik urutan array
count(x)	Menghitung jumlah elemen bernilai <i>x</i> pada array
index(x)	Membaca nilai pada elemen <i>x</i>
append(x)	Menulis ulang nilai pada index <i>x</i>

Tabel 3.1: Method pada modul array

3.3.2 List

List pada Python adalah rangkaian nilai-nilai yang dapat diakses menggunakan satu nama tunggal. List adalah container sekuensial yang dinamis, dapat menyimpan data dengan tipe data heterogen dan ukurannya dapat ditambah atau dikurangi secara dinamis. List juga bisa berisi list lainnya. Rangkaian nilai-nilai tersebut dituliskan di dalam [] seperti contoh berikut

```

1 nilai_ujian = [80,75,70,90,81,84,92,71,65,80,70]
2 nama_pahlawan = ['Sukarno', 'Diponegoro', 'Jend. Sudirman', 'Cut Nya Dhien']
3 nilai_campuran = ['Javascript', 20, 34.4, True]
4 list_dalam_list = [23, [22, 20], 45]

```

Operasi menambah

Terdapat Tiga metode (method) atau fungsi yang bisa digunakan untuk menambahkan isi atau item ke List:

1. prepend(item) : menambahkan item dari depan;
2. append(item) : menambahkan item dari belakang.
3. insert(index, item) : menambahkan item dari indeks tertentu

Contoh penggunaan adalah sebagai berikut :

```
1 #list mula-mula
2 buah = ["jeruk", "apel", "mangga", "duren"]
3 # Tambahkan manggis
4 buah.append("manggis")
5 buah.prepend("anggur")
6 buah.insert(2, "duren")
```

Operasi mengubah

List bersifat mutable, artinya isinya bisa kita ubah-ubah. Contoh mengubah list adalah sebagai berikut :

```
1 # list mula-mula
2 buah = ["jeruk", "apel", "mangga", "duren"]
3 # mengubah nilai index ke-2
4 buah[2] = "kelapa"
```

Operasi menghapus

Untuk menghapus salah satu isi dari List, kita bisa menggunakan perintah del. Contoh menghapus list adalah sebagai berikut :

```
1 # Membuat List
2 makul_list = [
3     "Pemrograman Android",
4     "PBO",
5     "Admin Basis Data",
6     "Jaringan Komputer",
7     "Struktur Data"
8 ]
9
10 # Misalkan kita ingin menghapus "Jaringan Komputer"
11 # yang berada di indeks ke-3
12 del makul_list[3]
13
14 print makul_list
```

Operasi menggabungkan

Dua list yang berbeda dapat digabungkan dengan menggunakan method extend()

```
1 pyListA = [ 12, 2 ]
2 pyListB = [ 79, 34, 17, 50, 18, 64 ]
3 pyListA.extend( pyListB )
4 print(pyListA)
```

3.3.3 Array 2 Dimensi

List dapat juga memiliki lebih dari satu dimensi atau disebut dengan multi dimensi. List multi dimensi biasanya digunakan untuk menyimpan struktur data yang kompleks seperti tabel, matriks, graph, tree, dsb.

kopi	susu	teh
jus apel	jus melon	jus apel
es kopi	es coklat	es jeruk

Tabel 3.2: Tabel Minuman

Contoh implementasinya adalah sebagai berikut:

```

1  # List minuman dengan 2 dimensi
2  list_minuman = [
3      ["Kopi", "Susu", "Teh"],
4      ["Jus Apel", "Jus Melon", "Jus Jeruk"],
5      ["Es Kopi", "Es Campur", "Es Teler"]
6  ]
7
8  # Cara mengakses list multidimensi
9  # misalkan kita ingin mengambil "es kopi"
10 print list_minuman[2][0]
```

Angka dua 2 pada kode di atas, menunjukan indeks list yang akan kita akses. Kemudian setelah dapat list-nya baru kita ambil isinya.

Hasil outputnya:

```

1  "Es Kopi"
```

Apabila kita ingin menampilkan semua isi dalam list multidimensi, Anda dapat menggunakan perulangan seperti berikut:

```

1  # List minuman dengan 2 dimensi
2  list_minuman = [
3      ["Kopi", "Susu", "Teh"],
4      ["Jus Apel", "Jus Melon", "Jus Jeruk"],
5      ["Es Kopi", "Es Campur", "Es Teler"]
6  ]
7
8  for menu in list_minuman:
9      for minuman in menu:
10         print minuman
```

```

1  Kopi
2  Susu
3  Teh
4  Jus Apel
5  Jus Melon
6  Jus Jeruk
7  Es Kopi
8  Es Campur
9  Es Teler
```

3.3.4 Sets

Tipe data Set Python adalah tipe data yang berisi kumpulan tipe data dan dipakai untuk mengolah himpunan (set). Jika dibandingkan dengan tipe data berbentuk array lain di Python, tipe data set berbeda dalam hal index, pengurutan dan keunikan nilai (unique value).

Tipe data set tidak memiliki index, sehingga tidak ada mekanisme pengurutan. Maksudnya, ketika kita menginput beberapa nilai ke dalam tipe data set, posisi nilai tersebut bisa berada di mana saja, tidak persis seperti apa yang tertulis. Akibat tidak memiliki index, maka kita tidak bisa menambah nilai baru ke dalam tipe data set dengan cara menulis nomor index (seperti di dalam tipe data list)

Ciri khas selanjutnya dari tipe data set adalah tidak bisa menerima nilai ganda, setiap nilai di dalam set harus unik. Jika kita menginput beberapa nilai yang sama, hanya satu yang tersimpan di dalam set.

Untuk membuat tipe data set, gunakan tanda kurung kurawal, kemudian setiap anggota set dipisah dengan tanda koma. Berikut contohnya:

```
1 foo = {100, 200, 300, 400}
2 bar = {"Python", 200, 6.99, True}
3
4 print(foo)
5 print(bar)
```

```
1 {200, 100, 400, 300}
2 {200, True, 6.99, 'Python'}
```

Selain perbedaan tanda kurung, cara penulisan tipe data set tidak berbeda dengan tipe data list yang sudah kita pelajari sebelum ini.

Perhatikan urutan data yang tampil, hampir semuanya tidak terurut sesuai penulisan. Ini karena di dalam tipe data set setiap anggota atau element tidak memiliki index sehingga posisinya tidak bisa dipastikan. Menggunakan function type(), kita bisa melihat perbedaan cara penulisan tipe data list, dan set dalam bahasa Python.

Operasi menambah

Untuk menambah elemen baru pada suatu himpunan kita menggunakan fungsi add().

```
1 nilai = set([3,6,9,2,5,7])
2
3 print("\nSebelum ditambah")
4 print (nilai)
5 #Menambah elemen baru dengan nilai 4 kedalam himpunan
6 nilai.add(4)
7 print("\nSetelah ditambah")
8 print (nilai)
```

Menghapus Elemen Set

Untuk menghapus elemen dari himpunan kita menggunakan metode remove(). Berikut contohnya:

```

1 nilai = set([3,6,9,2,5,7])
2
3 print("\nSebelum dihapus")
4 print (nilai)
5 #Menghapus 5 dari himpunan
6 nilai.remove(5)
7 print("\nSetelah dihapus")
8 print (nilai)

```

Jika kita ingin menghapus semua anggota/elemen himpunan bisa menggunakan fungsi clear().

Penggunaan Frozenset

kita juga bisa membuat himpunan menggunakan metode frozenset() dengan sifat immutable (tidak dapat berubah elemen Sets).

```

1 nilai = frozenset([3,6,9,2,5,7])
2
3 #menambah elemen
4 nilai.add(4)
5 #menghapus elemen
6 nilai.remove(3)
7
8 print(nilai)

```

Operasi Union & Intersection

Tipe data set pada dasarnya adalah tipe data khusus yang dipakai untuk operasi himpunan, seperti operasi gabungan (union), operasi irisan (intersection).

berikut contoh penggunaan tipe data set untuk operasi Sets:

```

1 foo = {1, 2, 3, 4, 5}
2 bar = {3, 4, 5, 6, 7}
3
4 print (foo | bar) # union
5 print (foo & bar) # intersection

```

```

1 {1, 2, 3, 4, 5, 6, 7}
2 {3, 4, 5}

```

Dalam contoh di atas, perintah `foo | bar` adalah operasi union sets. Hasilnya, seluruh anggota sets yang ada di variabel `foo` digabung dengan seluruh anggota dalam variabel `bar`, anggota yang ada di kedua himpunan hanya akan ditampilkan 1 kali.

Perintah `foo & bar` adalah operasi intersection sets. Hasilnya adalah seluruh anggota yang terdapat di variabel `foo` dan juga ada di variabel `bar` (harus ada di kedua variabel).

3.3.5 Tuple

Tipe data Tuple sangat mirip seperti tipe data List yang sudah kita pelajari sebelumnya, dimana tipe data Tuple juga terdiri dari kumpulan tipe data lain. Bedanya, anggota di dalam tipe data Tuple

tidak bisa diubah setelah di deklarasikan. Kita akan bahas perbedaan ini menggunakan contoh kode program.

Untuk membuat tipe data Tuple, gunakan tanda kurung biasa, kemudian setiap anggota list dipisah dengan tanda koma. Berikut contohnya:

```
1 foo = (100, 200, 300, 400)
2 bar = ("Python", 200, 6.99, True)
```

Nyaris tidak berbeda dengan List. Namun tipe data Tuple dibuat menggunakan tanda kurung bulat, bukan tanda kurung siku sebagaimana List.

Cara mengakses tipe data Tuple tidak berbeda dengan tipe data List, dimana kita menulis nomor urut index dalam tanda kurung siku:

```
1 foo = ("Python", 200, 6.99, True, 'Struktur Data', 99j)
2
3 print(foo[0])
4 print(foo[1])
5 print(foo[2])
6
7 #mengakses sekumpulan (range) elemen pada tuple
8 print(foo[2:5])
9 print(foo[:3])
10 print(foo[3:])
11 print(foo[:])
```

Hasil kode program python:

```
1 Python
2 200
3 6.99
4 (6.99, True, 'Struktur Data')
5 ('Python', 200, 6.99)
6 (True, 'Struktur Data', 99j)
7 ('Python', 200, 6.99, True, 'Struktur Data', 99j)
```

Saat tuple dibuat, maka nilai elemen dan jumlahnya tidak dapat dirubah (immutable). Benarkah demikian? mari kita coba:

```
1 foo = ("Python", 200, 6.99, True, 'Struktur Data', 99j)
2 foo[0] = 'Belajar'
3 print(foo)
```

Hasil kode program python:

```
1 Traceback (most recent call last):
2 File "D:\..\ListExample.py", line 3, in <module>
3 foo[0] = 'Belajar'
4 TypeError: 'tuple' object does not support item assignment
```

Hasilnya terjadi error dengan pesan: ‘tuple’ object does not support item assignment, yang kurang lebih berarti: “Tuple tidak mendukung penambahan nilai baru“. Inilah pembeda antara tipe data Tuple dengan tipe data List.

3.3.6 Dictionary

Tipe data dictionary adalah tipe data array dimana key atau index dari array bisa berbentuk string, tidak hanya number saja. Dalam bahasa pemrograman lain, dictionary ini dikenal juga dengan sebutan associative array. Tipe data dictionary ini cocok dipakai untuk kelompok data yang kompleks (terdiri dari banyak element).

Dalam pembuatan dictionary, kita menggunakan tanda kurung kurawal dan . Selain itu setiap element merupakan pasangan dari key dan value. Antar satu element dengan element lain dipisah dengan tanda koma seperti contoh berikut:

```

1 foo = { 1: "Belajar", 2: "Python", 3: "di Struktur Data" }
2 bar = { "mengapa": "Belajar", "apa": "Python", "dimana": "di Struktur Data" }
3 baz = { 1: "Belajar", "apa": "Python", "dimana": "di Struktur Data" }
4
5 print(type(foo))
6 print(type(bar))
7 print(type(baz))
8
9 print(foo)
10 print(bar)
11 print(baz)

```

Hasil kode program:

```

1 <class 'dict'>
2 <class 'dict'>
3 <class 'dict'>
4
5 {1: 'Belajar', 2: 'Python', 3: 'di Struktur Data'}
6 {'mengapa': 'Belajar', 'apa': 'Python', 'dimana': 'di Struktur Data'}
7 {1: 'Belajar', 'apa': 'Python', 'dimana': 'di Struktur Data'}

```

Untuk menampilkan satu element yang ada di dalam dictionary, tulis key atau index yang ingin diakses dalam kurung siku.

```

1 foo = { "kegiatan": "Praktikum Struktur Data Dengan Python",
2         "kelas": "Praktikum",
3         "hasil": "Yakin bisa!" }
4
5 print(foo["kegiatan"])

```

Hasil kode program:

```

1 Praktikum Struktur Data Dengan Python

```

Operasi mengubah element dictionary

Setelah di deklarasikan, kita bisa mengubah nilai dari sebuah element dictionary. Caranya mirip seperti tipe data list, yakni mengisi nilai ke dalam key atau index dictionary:

```
1 foo = { "kegiatan": "Praktikum Struktur Data Dengan Python",  
2         "kelas": "Praktikum",  
3         "hasil": "Yakin bisa!" }  
4  
5 foo["kelas"] = "Pembelajaran Jarak Jauh"  
6 print(foo)
```

```
1 {  
2     "kegiatan": "Praktikum Struktur Data Dengan Python",  
3     "kelas": "Pembelajaran Jarak Jauh",  
4     "hasil": "Yakin bisa!"  
5 }
```

Operasi menambah element dictionary

Untuk menambah element baru ke dalam dictionary, bisa dilakukan dengan cara mengisi nilai ke sebuah key baru:

```
1 foo = { "kegiatan": "Praktikum Struktur Data Dengan Python",  
2         "kelas": "Praktikum",  
3         "hasil": "Yakin bisa!" }  
4  
5 foo["target"] = 2020  
6 print(foo)
```

```
1 {  
2     'kegiatan': 'Praktikum Struktur Data Dengan Python',  
3     "kelas": 'Praktikum',  
4     'hasil': 'Yakin bisa!',  
5     'target': 2020  
6 }
```

Operasi menghapus element dictionary

Untuk menghapus element dari sebuah dictionary, bisa menggunakan perintah del. Berikut

```
1 foo = { "kegiatan": "Praktikum Struktur Data Dengan Python",  
2         "kelas": "Praktikum",  
3         "hasil": "Yakin bisa!" }  
4  
5 del foo["kelas"]  
6 print(foo)
```

```
1 {  
2     'kegiatan': 'Praktikum Struktur Data Dengan Python',  
3     'hasil': 'Yakin bisa!'  
4 }
```

Nested dictionary

Sebuah dictionary dapat diisi dengan banyak dictionary. Teknik ini disebut dengan nested dictionary.

```
1 people = {  
2     1: {'name': 'John', 'age': '27', 'sex': 'Male'},  
3     2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}  
4 }  
5 print(people)
```

```
1 {  
2     1: {'name': 'John', 'age': '27', 'sex': 'Male'},  
3     2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}  
4 }
```

Nested dictionary dapat diakses dengan cara sebagai berikut:

```
1 people = {  
2     1: {'name': 'John', 'age': '27', 'sex': 'Male'},  
3     2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}  
4 }  
5  
6 print(people[1]['name'])  
7 print(people[1]['age'])  
8 print(people[1]['sex'])
```

Outputnya adalah sebagai berikut :

```
1 John  
2 27  
3 Male
```

Nested dictionary dapat ditambah dengan cara sebagai berikut:

```
1 people = {  
2     1: {'name': 'John', 'age': '27', 'sex': 'Male'},  
3     2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}  
4 }  
5  
6 people[3] = {}  
7
```

```

8 people[3]['name'] = 'Luna'
9 people[3]['age'] = '24'
10 people[3]['sex'] = 'Female'
11 people[3]['married'] = 'No'
12
13 print(people[3])

```

Outputnya adalah sebagai berikut :

```

1 {'name': 'Luna', 'age': '24', 'sex': 'Female', 'married': 'No'}

```

Element pada nested dictionary dapat dihapus dengan cara sebagai berikut:

```

1 people = {1: {'name': 'John', 'age': '27', 'sex': 'Male'},
2           2: {'name': 'Marie', 'age': '22', 'sex': 'Female'},
3           3: {'name': 'Luna', 'age': '24', 'sex': 'Female', 'married': 'No'},
4           4: {'name': 'Peter', 'age': '29', 'sex': 'Male', 'married': 'Yes'}}
5
6 del people[3]['married']
7 del people[4]['married']
8
9 print(people[3])
10 print(people[4])

```

Outputnya adalah sebagai berikut :

```

1 {'name': 'Luna', 'age': '24', 'sex': 'Female'}
2 {'name': 'Peter', 'age': '29', 'sex': 'Male'}

```

Dalam program di atas, kita menghapus key:value "married" dari dictionary pada index 3 dan 4. Kemudian, kita mencetak people[3] dan people[4] untuk mengkonfirmasi perubahan.

Element pada nested dictionary dapat diakses menggunakan nested loop dengan cara sebagai berikut:

```

1 people = {1: {'Name': 'John', 'Age': '27', 'Sex': 'Male'},
2           2: {'Name': 'Marie', 'Age': '22', 'Sex': 'Female'}}
3
4 for p_id, p_info in people.items():
5     print("\nPerson ID:", p_id)
6
7     for key in p_info:
8         print(key + ': ', p_info[key])

```

Outputnya adalah sebagai berikut :

```

1 Person ID: 1
2 Name: John

```

```

3 Age: 27
4 Sex: Male
5
6 Person ID: 2
7 Name: Marie
8 Age: 22
9 Sex: Female

```

Dalam program di atas, loop pertama mengembalikan semua key dalam nested dictionary people. Ini terdiri dari ID p_id setiap orang. Kita menggunakan ID ini untuk membaca informasi p_info setiap people.

Looping kedua melewati informasi dari setiap people. Kemudian, proses akan mengembalikan semua key yaitu name, age, sex dictionary setiap people.

Dan terakhir, kita mencetak key dari people dan value untuk setiap key.

3.4 Guided

Pada bagian ini akan diberikan beberapa contoh program yang harus Anda coba untuk mendapatkan pemahaman lebih baik tentang materi pada bab ini.

Menghitung rata-rata dari nilai yang iniputkan

Buatlah sebuah program yang menghitung rata-rata dari nomer yang diinputkan.

Pembahasan

Langkah-langkah solusi untuk program diatas adalah sebagai berikut :

- Ambil jumlah angka yang akan diinputkan dan buatkan object collectionnya.
- Gunakan looping untuk meng-inputkan element satu-persatu pada collection
- Hitung nilai total dari collection.
- Lakukan pembagian nilai total dengan jumlah elemen yang ada pada collection.

Kode lengkapnya dalam Python adalah sebagai berikut:

```

1 n=int(input("Masukan jumlah elemen yang akan diinputkan: "))
2 a=[]
3 for i in range(0,n):
4     elem=int(input("Masukan element: "))
5     a.append(elem)
6 avg=sum(a)/n
7 print("Rata-rata elements pada list adalah ",round(avg,2))

```

Menghitung rata-rata dari nilai yang iniputkan

Buatlah sebuah program yang akan mengecek ada atau tidaknya key yang diinputkan pada dictionary.

Pembahasan

Langkah-langkah solusi untuk program diatas adalah sebagai berikut :

- Deklarasikan dan inisialisasikan sebuah dictionary dengan sebuah nilai
- Dapatkan input key dari user, dan simpan nilainya pada variable
- Gunakan statement if dan operator in, untuk memeriksa apakah key ada pada dictionary atau tidak.
- Jika key ditemukan, maka tampilkan nilainya.

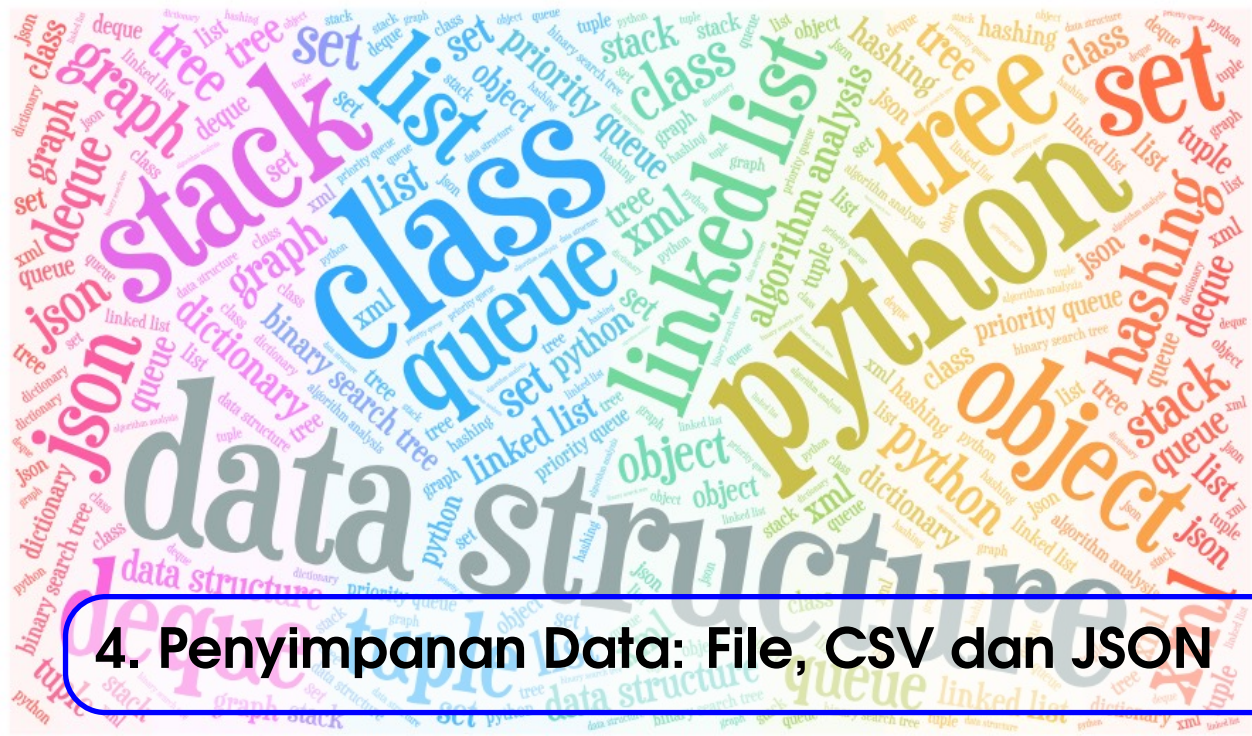
- Jika key tidak ditemukan, maka tampilkan pesan bahwa key tidak ditemukan.

Kode lengkapnya dalam Python adalah sebagai berikut:

```
1     d={'A':1, 'B':2, 'C':3}
2     key=input("Masukan key yang ingin dicari:")
3     if key in d.keys():
4         print("Key ditemukan, nilainya adalah :")
5         print(d[key])
6     else:
7         print("Key tidak ditemukan!")
```

3.5 Unguided

Bagian unguided akan diberikan secara terpisah pada saat praktikum di Lab.



4.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat melakukan operasi membaca dan menulis data pada file.
2. Dapat membaca data dan menyimpan data dalam format CSV.
3. Dapat membaca data dan menyimpan data dalam format JSON.

4.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.8, 3.9, atau 3.10 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).
6. Moodle UKDW

4.3 Materi

Membuka dan Menutup File Teks

Untuk membuka file dapat menggunakan fungsi `open()` dengan parameter `path` dari file yang akan dibuka dan mode pembacaannya. Setelah file berhasil dibuka, dapat dilanjutkan dengan operasi pembacaan isi maupun penulisan data baru ke dalam file. Berikut ini adalah contoh source code untuk membuka file, menulis data ke dalamnya dan menutup file.

```
1
2     fruits = open('fruits.txt', 'w')
3     fruits.write('Apple\n')
4     fruits.write('Mango\n')
5     fruits.write('Orange\n')
6     fruits.write('Banana\n')
7     fruits.close()
8
9     fruits = open('fruits.txt', 'r')
10    fruit_list = fruits.readlines()
11    print('Fruit count: ', len(fruit_list))
12    print(fruit_list)
13    fruits.close()
14
```

Penjelasan dari source code tersebut adalah sebagai berikut:

- Baris 1: Membuka file bernama fruits.txt yang berada di folder yang sama dengan file Python yang sedang dijalankan. Parameter 'w' berarti membuka file tersebut dalam mode write. Mode ini akan melakukan overwrite isi file.
- Baris 2-5: Menulis 4 baris data ke dalam file, setiap barisnya diakhiri dengan karakter newline.
- Baris 6: Menutup file.
- Baris 8: Membuka file kembali, dengan mode 'r' yang berarti read only.
- Baris 9: Membaca seluruh isi file dalam bentuk list.
- Baris 10: Menampilkan jumlah baris, sesuai dengan jumlah isi di dalam list.
- Baris 11: Menampilkan list
- Baris 12: Menutup file.

Output yang dihasilkan oleh source code tersebut adalah sebagai berikut:

```
Fruit count:  4
['Apple\n', 'Mango\n', 'Orange\n', 'Banana\n']
```

Menambah Isi File Teks (Append)

Pada contoh sebelumnya dilakukan pembukaan file dengan mode 'w' yang artinya write mode. Mode ini akan melakukan overwrite isi file, sehingga isi file sebelumnya akan dihapus dan digantikan dengan data yang ditulis. Pada beberapa kasus, bisa saja dibutuhkan untuk menambah isi file, sehingga akan terus bertambah isinya dan tidak terjadi overwrite. Untuk mencapai tujuan tersebut, gunakan parameter 'a' atau 'a+' saat membuka file. Berikut ini adalah contoh source code untuk menambah isi file berdasarkan input yang diberikan oleh pengguna program:

```
1
2     fruits = open('fruits.txt', 'a+')
3     fruits.seek(0)
4     fruit_list = fruits.readlines()
5     print('Fruit count: ', len(fruit_list))
6
7     n = int(input('How many fruits: '))
```

```

8     for i in range(0, n):
9         fruit = input('Enter new fruit: ')
10        fruits.write(fruit + '\n')
11        fruits.seek(0)
12
13        fruit_list = fruits.readlines()
14        print('Fruit count: ', len(fruit_list))
15        print(fruit_list)
16        fruits.close()
17

```

Penjelasan dari source code tersebut adalah sebagai berikut:

- Baris 1: Membuka file fruits.txt dengan mode append dan read ('a+'). Mode yang digunakan adalah 'a+' karena akan dilakukan pembacaan isi file terlebih dahulu sebelum menambah data ke dalamnya.
- Baris 2: Menggeser cursor ke awal file. Karena file dibuka dengan mode append, maka secara otomatis cursor diletakkan di akhir file. Karena itu perlu digeser terlebih dahulu ke depan sebelum melakukan operasi pembacaan.
- Baris 3: Membaca isi file ke dalam list bernama fruit_list.
- Baris 4: Menampilkan jumlah elemen di dalam fruit_list. Ini sama saja dengan jumlah baris di dalam file fruits.txt.
- Baris 6-7: Meminta input dari pengguna berupa jumlah data (n) yang akan ditambahkan ke dalam file.
- Baris 8: Secara berulang-ulang sebanyak n kali, minta pengguna memasukkan data baru.
- Baris 9: Tulis input pengguna ke dalam file, secara otomatis akan diletakkan di bagian akhir file karena file dibuka dalam mode append dan read.
- Baris 10: Setelah data baru berhasil ditambahkan, kembalikan cursor ke awal file karena akan dilakukan pembacaan isi file kembali.
- Baris 12-14: Menampilkan isi file setelah ditambah.
- Baris 15: Menutup file.

Output yang dihasilkan oleh source code tersebut adalah sebagai berikut:

```

Fruit count:  4
How many fruits: 3
Enter new fruit: Durian
Enter new fruit: Guava
Enter new fruit: Watermelon
Fruit count:  7
['Apple\n', 'Mango\n', 'Orange\n', 'Banana\n', 'Durian\n', 'Guava\n', 'Watermelon\n']

```

Menghapus Whitespace Dari Isi File Teks

Pada saat membaca file teks, setiap barisnya akan diakhiri dengan karakter newline, yang termasuk ke dalam jenis whitespace. Jika data di dalam file teks akan diproses atau diolah, maka whitespace yang ada perlu dihilangkan terlebih dahulu. Untuk menghilangkan whitespace dapat menggunakan fungsi strip() dari str. Berikut ini adalah contoh source code untuk membaca isi file dan menghapus semua whitespace yang ada di setiap barisnya:

```

1
2     fruits = open('fruits.txt', 'r')

```

```

3     fruit_list = fruits.readlines()
4     print('Fruit count: ', len(fruit_list))
5     print(fruit_list)
6     for fruit in fruit_list:
7         print(fruit.strip())
8     fruits.close()
9

```

Penjelasan dari source code tersebut adalah sebagai berikut:

- Baris 1: Membuka file fruits.txt dalam mode read.
- Baris 2: Membaca isi file ke dalam list fruit_list.
- Baris 3: Menampilkan jumlah elemen di dalam list fruit_list. Ini sama saja dengan jumlah baris di dalam file fruits.txt.
- Baris 4: Menampilkan isi dari list fruit_list. Setiap elemennya masih diakhiri dengan karakter whitespace (newline)
- Baris 5-6: Untuk setiap elemen di dalam list fruit_list, lakukan penghapusan karakter whitespace dengan pemanggilan fungsi strip()
- Baris 7: Menutup file.

Output yang dihasilkan oleh source code tersebut adalah sebagai berikut:

```

Fruit count:  7
['Apple\n', 'Mango\n', 'Orange\n', 'Banana\n', 'Durian\n', 'Guava\n', 'Watermelon\n']
Apple
Mango
Orange
Banana
Durian
Guava
Watermelon

```

Membaca Isi File CSV

File Comma-Separated Values adalah file yang berisi data yang umumnya dipisahkan menggunakan tanda ',' (koma). Walau namanya adalah Comma, tetapi tanda pemisah bisa saja menggunakan karakter lainnya. Karakter yang umumnya digunakan sebagai pemisah adalah: koma (,), titik koma(;), spasi (), sharp (#), percent (%), Tab dan beberapa karakter lainnya.

Membaca isi file CSV secara umum sama seperti membaca isi file teks. Hanya bedanya pada jumlah kolom data yang bervariasi di setiap barisnya. Pada baris pertama dari file CSV umumnya berisi keterangan judul dari setiap kolom data di dalam file. Contoh isi file CSV adalah sebagai berikut:

```

tahun,pria,wanita
2020,110,89
2019,90,102
2018,100,82
2017,89,91
2016,94,92

```

Baris pertama dari file tersebut adalah judul kolom data, yaitu tahun, pria dan wanita. Berarti untuk setiap barisnya, terdapat tiga kolom data. Baris kedua sampai baris terakhir berisi data. Untuk membaca isi file CSV tersebut, dapat digunakan source code seperti berikut ini:

```
1
2     print('Data mahasiswa berdasarkan angkatan dan jenis kelamin')
3     with open('mahasiswa.csv', 'r') as datafile:
4         list_data = datafile.readlines()
5         for i in range(1, len(list_data)):
6             data = list_data[i].strip()
7             split_data = data.split(',')
8             print('Angkatan ', split_data[0])
9             print('Pria: ', split_data[1])
10            print('Wanita: ', split_data[2])
11            print()
12
```

Penjelasan dari source code tersebut adalah sebagai berikut:

- Baris 1: Menampilkan tulisan.
- Baris 2: Membuka file menggunakan with. Dengan cara ini, berarti tidak perlu menutup file, karena akan ditutup secara otomatis setelah blok with berakhir. File yang dibuka bernama mahasiswa.csv yang dibuka dengan mode read. File tersebut akan dikenali sebagai variabel datafile di dalam blok with.
- Baris 3: Baca isi file ke dalam list list_data.
- Baris 4: Lakukan perulangan sejumlah elemen di dalam list_data, dimulai dari baris kedua. Ingat, baris pertama berisi judul kolom sehingga dilewati. Data dimulai sejak baris kedua (index ke-1).
- Baris 5: Baca elemen ke-i dari list_data dan hapus semua whitespace yang ada.
- Baris 6: Lakukan split berdasarkan tanda koma. Hasilnya disimpan dalam list split_data.
- Baris 7-10: Tampilkan isi data berdasarkan hasil fungsi split().

Output yang dihasilkan oleh source code tersebut adalah sebagai berikut:

```
Data mahasiswa berdasarkan angkatan dan jenis kelamin
```

```
Angkatan  2020
```

```
Pria:  110
```

```
Wanita:  89
```

```
Angkatan  2019
```

```
Pria:  90
```

```
Wanita:  102
```

```
Angkatan  2018
```

```
Pria:  100
```

```
Wanita:  82
```

```
Angkatan  2017
```

```
Pria:  89
```

```
Wanita:  91
```

```
Angkatan 2016
Pria: 94
Wanita: 92
```

4.3.1 Membaca Isi File CSV Dengan Modul csv

Python menyediakan modul csv yang dapat digunakan untuk membaca maupun menulis file csv dengan cara yang relatif lebih mudah jika dibandingkan dengan penggunaan pembacaan file pada contoh sebelumnya. Berikut ini adalah source code yang menggunakan modul csv untuk membaca file mahasiswa.csv:

```
1
2 import csv
3 print('Data mahasiswa berdasarkan angkatan dan jenis kelamin')
4 with open('mahasiswa.csv', 'r', newline='') as datafile:
5     reader = csv.DictReader(datafile)
6     for row in reader:
7         print('Angkatan ', row['tahun'])
8         print('Pria: ', row['pria'])
9         print('Wanita: ', row['wanita'])
10    print()
11
```

Penjelasan dari source code tersebut adalah sebagai berikut:

- Baris 1: Import modul csv.
- Baris 2: Menampilkan tulisan.
- Baris 3: Membuka file mahasiswa.csv dengan mode read. File tersebut akan dikenali sebagai datafile di dalam blok with. Jika menggunakan modul csv, maka pada pembukaan file perlu ditambahkan parameter `newline=""`. Parameter tersebut diperlukan karena seluruh karakter `newline` di dalam file csv akan ditangani langsung oleh modul csv.
- Baris 4: Baris pertama file mahasiswa.csv berisi judul kolom data, karena itu data di dalam file tersebut dapat dibaca dalam bentuk dictionary. Untuk membaca data dalam bentuk dictionary, gunakan fungsi `DictReader()` dari modul csv.
- Baris 5: Lakukan pembacaan per-baris data.
- Baris 6-9: Akses data dengan menggunakan model akses seperti dictionary. Anda perlu menuliskan nama kolom untuk mendapatkan datanya.

Output yang dihasilkan oleh source code tersebut adalah sebagai berikut:

```
Data mahasiswa berdasarkan angkatan dan jenis kelamin
Angkatan 2020
Pria: 110
Wanita: 89

Angkatan 2019
Pria: 90
Wanita: 102
```



```
Angkatan 2018
Pria: 100
Wanita: 82
```

```
Angkatan 2017
Pria: 89
Wanita: 91
```

```
Angkatan 2016
Pria: 94
Wanita: 92
```

Menyimpan Data Dalam Format JSON

JSON merupakan singkatan dari Javascript Object Notation. Sesuai dengan namanya, format data ini berasal dari bahasa pemrograman Javascript. Pada saat ini, hampir semua bahasa pemrograman mendukung format data ini. Format JSON adalah salah satu format data yang paling banyak digunakan untuk pertukaran data melalui API. Python sudah mendukung JSON secara native dan sudah disesuaikan dengan tipe data yang ada di Python.

Proses penyimpanan data dari memory menjadi file dalam format JSON sering disebut sebagai proses serialisasi. Pada proses serialisasi terjadi konversi tipe data Python menjadi tipe data yang dikenali oleh JSON. Konversi dilakukan agar isi file JSON sesuai dengan standar dan dapat dibaca oleh bahasa pemrograman lainnya. Untuk melakukan serialisasi dapat menggunakan fungsi `dump()` dari modul `json`, seperti pada contoh berikut ini:

```
1
2 import json
3 agus = dict()
4 agus['norekening'] = 8199918188
5 agus['nama'] = 'Agus Bagus'
6 agus['saldo'] = '19.57'
7 agus['matauang'] = 'USD'
8
9 bambang = dict()
10 bambang['norekening'] = 9288818881
11 bambang['nama'] = 'Bambang Gentolet'
12 bambang['saldo'] = '123000'
13 bambang['matauang'] = 'Yen'
14
15 data = [agus, bambang]
16
17 with open('nasabah.json', 'w') as datafile:
18     json.dump(data, datafile)
19
```

Penjelasan dari source code tersebut adalah sebagai berikut:

- Baris 1: Import modul `json`.
- Baris 2-6: Buat sebuah dict berisi beberapa data.
- Baris 8-12: Buat sebuah dict lagi yang berisi beberapa data.

- Baris 14: Buat sebuah list yang berisi kedua dict tersebut.
- Baris 16-17: Simpan list tersebut ke dalam file `nasabah.json`, dalam format JSON.

Jika source code tersebut dijalankan, akan dihasilkan file `nasabah.json` seperti berikut ini:

```
[
{
  "norekening": 8199918188,
  "nama": "Agus Bagus",
  "saldo": "19.57",
  "matauang": "USD"
},
{
  "norekening": 9288818881,
  "nama": "Bambang Gentolet",
  "saldo": "123000",
  "matauang": "Yen"
}
]
```

Membaca Data Di Dalam File JSON

Pembacaan data dari file JSON dinamakan deserialisasi. Proses ini merupakan kebalikan dari serialisasi. Pada deserialisasi, tipe data pada JSON diubah menjadi tipe data pada Python. Untuk melakukan deserialisasi dapat menggunakan fungsi `load()` dari modul `json`, seperti pada contoh berikut ini:

```
1
2 import json
3 with open('nasabah.json', 'r') as datafile:
4     data = json.load(datafile)
5     print('Daftar nasabah: ')
6     for nasabah in data:
7         print(nasabah['nama'], ' - ', nasabah['norekening'])
8         print('Saldo: ', nasabah['saldo'])
9         print('Mata uang: ', nasabah['matauang'])
10
```

Penjelasan dari source code tersebut adalah sebagai berikut:

- Baris 1: Import modul `json`.
- Baris 2: Membuka file `nasabah.json` dengan mode `read`. File tersebut akan dikenali dengan nama `datafile` di dalam blok `with`. Dengan menggunakan `with` berarti tidak perlu menutup file, karena akan dilakukan secara otomatis setelah blok `with` berakhir.
- Baris 3: Lakukan deserialisasi dengan fungsi `load()` dari modul `json`. Hasilnya disimpan pada variabel `data`. Pada contoh ini, variabel `data` akan berupa list yang di dalamnya berisi dua dictionary.
- Baris 4: Menampilkan tulisan.
- Baris 5: Untuk setiap dict di dalam list `data`, tampilkan isinya satu-persatu.
- Baris 6-8: Menampilkan data setiap nasabah.

Output yang dihasilkan oleh source code tersebut adalah sebagai berikut:

```
Daftar nasabah:  
Agus Bagus - 8199918188  
Saldo: 19.57  
Mata uang: USD  
Bambang Gentolet - 9288818881  
Saldo: 123000  
Mata uang: Yen
```

4.4 Guided

Mengubah Isi File JSON

Lakukan perubahan pada file nasabah.json sebagai berikut:

- Ubah saldo Bambang Gentolet menjadi 300000
- Ubah nama Agus Bagus menjadi Agus Gus Gus Gus

Pembahasan Mengubah Isi File JSON

Untuk melakukan perubahan pada isi file nasabah.json, maka langkah-langkah yang perlu dilakukan adalah sebagai berikut:

- Buka file nasabah.json.
- Lakukan deserialisasi, sehingga sudah menjadi tipe data pada Python.
- Lakukan perubahan nilai sesuai yang diminta.
- Simpan kembali ke dalam file nasabah.json (serialisasi).

Langkah-langkah tersebut dapat diwujudkan dalam bentuk source code berikut ini:

```
1  
2     import json  
3     data = None  
4     with open('nasabah.json', 'r') as datafile:  
5         data = json.load(datafile)  
6  
7     data[0]['nama'] = 'Agus Gus Gus Gus'  
8  
9     data[1]['saldo'] = 300000  
10  
11    with open('nasabah.json', 'w') as datafile:  
12        json.dump(data, datafile)  
13
```

Penjelasan dari source code tersebut adalah sebagai berikut:

- Baris 1: Import modul json.
- Baris 2: Sebelum membaca file nasabah.json, siapkan variabel data yang akan digunakan untuk menampung hasil deserialisasi.
- Baris 3: Membuka file nasabah.json dalam mode read, file akan dikenali dengan nama datafile di dalam blok with. Karena menggunakan with, maka tidak perlu menutup file karena secara otomatis file akan ditutup setelah blok with berakhir.

- Baris 4: Lakukan deserialisasi dari `nasabah.json`, hasilnya dimasukkan ke variabel data. Variabel data sekarang sudah berisi list yang di dalamnya terdapat dua dict.
- Baris 6, 8: Lakukan perubahan nilai sesuai permintaan.
- Baris 10-11: Simpan perubahan ke file `nasabah.json` dengan cara melakukan serialisasi dari variabel data.

Menambah Data Dalam File CSV

Jika anda diberi file `menu.csv` yang berisi daftar menu di suatu warung makan seperti berikut ini:

```
Makanan;Ayam Goreng;15000
Makanan;Sate Ayam;15000
Minuman;Es Teh;8000
Minuman;Es Jeruk;10000
Makanan;Nasi Putih;4000
Makanan;Ayam Bakar;17000
```

Buatlah sebuah file `menu-new.csv` yang isinya sama dengan `menu.csv` dengan tambahan menu berikut ini:

- Makanan;Ayam Geprek Lombok Ijo;20000
- Makanan;Bebek Goreng;25000
- Makanan;Sate Kambing;25000
- Minuman;Es Doger;15000

Sehingga isi file `menu-new.csv` akan seperti berikut ini:

```
Makanan;Ayam Goreng;15000
Makanan;Sate Ayam;15000
Minuman;Es Teh;8000
Minuman;Es Jeruk;10000
Makanan;Nasi Putih;4000
Makanan;Ayam Bakar;17000
Makanan;Ayam Geprek Lombok Ijo;20000
Makanan;Bebek Goreng;25000
Makanan;Sate Kambing;25000
Minuman;Es Doger;15000
```

Pembahasan Menambah Data Dalam File CSV

Untuk menghasilkan file `menu-new.csv` sesuai yang diminta, maka langkah-langkah yang perlu dilakukan adalah sebagai berikut:

- Buka dan Baca isi file `menu.csv`. Simpan hasilnya dalam sebuah list. Tutup file setelah pembacaan berhasil.
- Tambahkan data baru ke dalam list.
- Simpan isi list ke file `menu-new.csv`. Tutup file setelah penulisan berhasil.

Langkah-langkah tersebut dapat diwujudkan dalam bentuk source code berikut ini:

```
1
2 # baca isi file menu.csv
```

```

3     data = None
4     with open('menu.csv', 'r') as datafile:
5         data = datafile.readlines()
6         for i in range(0, len(data)):
7             data[i] = data[i].strip()
8
9     # tambahkan data baru
10    data.append('Makanan;Ayam Geprek Lombok Ijo;20000')
11    data.append('Makanan;Bebek Goreng;25000')
12    data.append('Makanan;Sate Kambing;25000')
13    data.append('Minuman;Es Doger;15000')
14
15    # simpan dalam file menu-new.csv
16    with open('menu-new.csv', 'w') as datafile:
17        for line in data:
18            datafile.write(line + '\n')
19

```

Pada baris 5-6 dilakukan operasi strip() untuk menghilangkan semua whitespace yang ada. Pada saat penulisan ke file menu-new.csv tambahkan newline seperti pada baris 17.

Pengolahan Data Dari File CSV

Dari file menu-new.csv, hitunglah total harga makanan dan harga minuman yang ada! Pada kasus ini anda berarti diminta membaca data dari file menu-new.csv, kemudian mengelompokkan menjadi 2, yaitu makanan dan minuman. Kemudian jumlahkan semua harga minuman untuk mendapatkan total harga minuman. Jumlahkan semua harga makanan untuk mendapatkan total harga makanan.

Pembahasan Pengolahan Data Dari File CSV

Untuk menyelesaikan masalah ini, langkah-langkah yang perlu dilakukan adalah sebagai berikut:

- Buka dan baca isi file menu-new.csv.
- Kelompokkan menjadi dua list, masing-masing berisi harga makanan dan harga minuman.
- Hitung jumlah total harga makanan dan minuman. Tampilkan hasilnya di layar.

Langkah-langkah tersebut dapat diwujudkan dalam bentuk source code berikut ini:

```

1
2     import csv
3
4     # siapkan list harga minuman dan makanan
5     makanan = []
6     minuman = []
7
8     # baca isi file menu-new.csv dengan DictReader
9     with open('menu-new.csv', 'r', newline='') as datafile:
10        # buat nama kolom
11        fields = ['Jenis', 'Nama', 'Harga']
12        reader = csv.DictReader(datafile, fieldnames=fields, delimiter=';')
13
14        # kelompokkan harga makanan dan minuman

```

```
15     for line in reader:
16         if line['Jenis'] == 'Makanan':
17             makanan.append(int(line['Harga']))
18         elif line['Jenis'] == 'Minuman':
19             minuman.append(int(line['Harga']))
20
21     # hitung total harga makanan
22     total_makanan = sum(makanan)
23     total_minuman = sum(minuman)
24
25     # tampilkan hasil
26     print('Total makanan: ', total_makanan)
27     print('Total minuman: ', total_minuman)
28
```

Untuk membaca csv menggunakan DictReader diperlukan judul kolom, sehingga definisikan dulu judul kolom di baris 10. Kemudian bacalah isinya dalam bentuk dict dengan menggunakan DictReader() seperti pada baris 11. Gunakan parameter fieldnames untuk mengatur nama kolom, sedangkan parameter delimiter digunakan untuk memberitahu delimiter yang digunakan. Karena file menu-new.csv menggunakan delimiter titik koma (;) maka perlu diberitahu pada saat membuat DictReader.

Jika source code tersebut dijalankan, berikut ini adalah output yang dihasilkan:

```
Total makanan: 121000
Total minuman: 33000
```

4.5 Unguided

Bagian unguided akan diberikan secara terpisah pada saat praktikum di Lab.

5.3 Materi

5.3.1 Paradigma Pemrograman

Paradigma pemrograman bisa diartikan sebagai cara pandang penyelesaian suatu masalah komputasi dalam pemrograman komputer. Paradigma pemrograman biasanya berupa cara untuk mengklasifikasikan bahasa pemrograman berdasarkan fitur mereka. Terdapat beberapa klasifikasi paradigma pemrograman, yaitu:


1. **Pemrograman Terstruktur** Pada pemrograman terstruktur, penyelesaian masalah komputasi dilakukan secara struktural, terurut, dari awal hingga akhir. Pemrograman terstruktur ini biasanya disebut juga pemrograman prosedural. Pada pemrograman prosedural, pengguna atau user harus memberikan serangkaian perintah yang berurutan untuk menyelesaikan suatu masalah pemrograman. Bila perintah tidak dilakukan secara terurut, masalah yang muncul kemungkinan tidak dapat terselesaikan dengan baik. Pemrograman model ini sudah dipelajari pada mata kuliah Algoritma Pemrograman yang menggunakan fungsi-fungsi / prosedur-prosedur terpisah untuk menyelesaikan sub masalah sedikit demi sedikit hingga akhirnya semua masalah secara utuh terselesaikan. Di dalam model ini semua fungsi / prosedur dieksekusi secara urut. Ciri-ciri:
 - Adanya bentuk looping (pengulangan)
 - Unit abstraksi berupa prosedur dan fungsi (sub program)
 - Step program intinya menerangkan bagaimana caranya menyelesaikan problem (how)
2. **Pemrograman Berorientasi Obyek** Lain halnya dengan sebelumnya, paradigma pemrograman berorientasi objek menyelesaikan suatu masalah yang ada dengan berorientasi kepada objek, karena seluruh data serta fungsi yang ada di dalamnya dikemas dalam suatu kelas (class) atau objek-objek yang terpisah ke beberapa bagian tertentu. Hal ini sangat berbeda dengan pemrograman prosedural atau iteratif yang sudah dijelaskan sebelumnya karena setiap objek yang ada dapat menerima pesan yang dikirim, memproses data yang ada, serta mengirimkan pesan ke objek lainnya tanpa harus melakukannya secara berurut karena dapat dilakukan sekaligus dalam satu waktu. Pada struktur data, class dan objek digunakan sebagai kerangka pemrograman saja, tidak sampai dengan konsep OOP yang dalam. Ciri: unit abstraksi berupa class/object.

5.3.2 Class

Class merupakan template/blueprint dari object - object. Object diciptakan dari class, sehingga akan memiliki karakteristik yang sama dengan class tersebut. Jika ada beberapa object diciptakan dari Class yang sama, maka object-object tersebut akan memiliki karakteristik dan perilaku yang sama. Secara umum, sebuah class terdiri dari beberapa bagian berikut:

- Atribut/State.
- Constructor.
- Method/Behavior.

Contoh class adalah: class Mobil, class Lampu, class KartuKredit, class Baju, class Manusia, class Buku dan lain-lain.

 Class perlu dibuat dulu, kemudian baru dapat membuat Object berdasarkan Class tersebut.

Contoh pada Python untuk membuat class Mahasiswa dengan 3 atribut: nim, nama, dan ipk

```
1 class Mahasiswa:
2     _nim=""
3     _nama=""
4     _ipk=0.0
```


5.3.3 Object

Object adalah suatu bentuk nyata dari sebuah class. Object harus diciptakan dari sebuah class. Object menggambarkan kondisi nyata dari suatu class dalam memori komputer. Jika class dibuat menjadi beberapa object, maka masing-masing object tersebut memiliki nilai masing-masing yang bisa berbeda-beda atau sama dari atribut class tersebut.

Contoh object adalah: class Mobil - object BMW, class Lampu - object Neon, class KartuKredit - object CCBCA, class Baju - object Hem, class Manusia - object Mahasiswa, class Buku - object Novel, dan lain-lain.

Contoh pembuatan object pada Python dari class Mahasiswa:

```
1 m = Mahasiswa
2 m._nim="1"
3 print(m._nim)
```

Hasil adalah 1

5.3.4 Atribut

Atribut adalah sesuatu yang dimiliki dan melekat pada setiap object yang diciptakan dari class. Pada pendefinisian class terdapat bagian atribut yang menggambarkan variabel class yang melekat pada class tersebut. Atribut sebaiknya dilindungi dari pengaksesan dari luar class yang tidak seharusnya dilakukan. Contoh atribut dari class Mahasiswa adalah nim, nama, ipk, atribut dari class KartuKredit adalah nomor_kartu, nama_pemilik, limit, saldo, atribut dari class Novel adalah isbn, judul, pengarang, harga, tebal, dan ukuran. Atribut sering disebut dengan variabel class.

Contoh pada Python:

```
1 class KartuKredit:
2     _noKartu=""
3     _namaPengguna=""
4     _limit=0
5     _saldo=0
6
```

5.3.5 Method

Method adalah tingkah laku dari suatu object yang diciptakan dari sebuah class. Pada pendefinisian class terdapat bagian untuk method yang diimplementasikan dalam bentuk fungsi / prosedur. Fungsi/prosedur tersebut biasanya terdiri dari 2 jenis, yaitu method yang disebut mutator (yang bersifat mengubah nilai atribut) dan asesor (yang bersifat membaca nilai atribut).

Contoh method yang berhubungan dengan atribut class KartuKredit adalah getNomorKartu, setNomorKartu, getNama, setNama, getLimit, setLimit, setSaldo dan getSaldo. Sedangkan untuk method yang tidak langsung berhubungan dengan atribut adalah transaksi() yang akan mengurangi limit dan saldo dari KartuKredit.

Contoh pada Python:

```
1 class KartuKredit:
2     _noKartu=""
```

```

3     _namaPengguna=""
4     _limit=0.0
5     _saldo=0.0
6
7     def setNoKartu(self,noKartu):
8         self._noKartu = noKartu
9
10    def setNamaPengguna(self,nama):
11        self._namaPengguna = nama
12
13    def getNoKartu(self):
14        return self._noKartu
15
16    def getNamaPengguna(self):
17        return self._namaPengguna
18
19    #cara penggunaan:
20    cc = KartuKredit()
21    cc.setNoKartu("1")
22    cc.setNamaPengguna("anton")
23    print(cc.getNamaPengguna())
24    print(cc.getNoKartu())

```

Pada Python untuk membuat method digunakan fungsi / procedure, yaitu menggunakan keyword `def`. Keyword yang digunakan sebagai parameter pada fungsi adalah `self`. Keyword `self` menggambarkan class itu sendiri dan kemudian di dalam method digunakan untuk mengakses atributnya (variabel class).

Fungsi berjenis asesor menggunakan format nama `getXXX`, misalnya `getNoKartu`, sedangkan fungsi berjenis mutator menggunakan format nama `setXXX`, misalnya `setNoKartu`. Kegunaan `setXXX` adalah untuk mengisi / mengubah nilai variabel class / atribut, sedangkan kegunaan `getXXX` adalah untuk mengambil nilai variabel class / atribut.

5.3.6 Konstruktor

Konstruktor adalah bagian dari class yang mirip method yang dipakai dan digunakan untuk menginstansiasi sebuah class. Menginstansiasi adalah mempersiapkan memory dari class itu untuk digunakan. Biasanya saat instansiasi juga akan dilakukan pemanggilan konstruktor tersebut. Konstruktor biasanya digunakan untuk memberi nilai awal (inisialisasi) dari atribut class.

```

1 class KartuKredit:
2     _noKartu=""
3     _namaPengguna=""
4     _limit=0.0
5     _saldo=0.0
6
7     def __init__(self,noKartu,namaPengguna,limit,saldo):
8         self._noKartu = noKartu
9         self._namaPengguna = namaPengguna
10        self._limit = limit

```

```

11         self._saldo = saldo
12

```

Pada Python konstruktor menggunakan keyword `__init__`. Method ini diikuti dengan parameter `self` dan semua variabel-variabel input untuk menginisialisasi semua variabel class yang dimaksud. Dengan menggunakan konstruktor ini maka semua variabel class akan diisi dengan nilai awalnya.

```

1 class KartuKredit:
2     _noKartu=""
3     _namaPengguna=""
4     _limit=0.0
5     _saldo=0.0
6
7     def __init__(self,noKartu,namaPenguna,limit,saldo):
8         self._noKartu = noKartu
9         self._namaPengguna = namaPenguna
10        self._limit = limit
11        self._saldo = saldo
12
13    #cara penggunaan:
14    cc = KartuKredit("1","anton",10000,500)
15    #method di bawah ini dianggap ada
16    print(cc.getNamaPengguna())
17    print(cc.getNoKartu())
18    print(cc.getLimit())
19    print(cc.getSaldo())

```

5.4 Guided

Pada bagian ini akan diberikan beberapa latihan soal dan sekaligus pembahasan soal-soal tersebut.

Lingkaran

Buatlah class `Lingkaran` yang menggambarkan bentuk benda 2 dimensi berupa lingkaran. Atribut lingkaran adalah jari-jari serta memiliki method `hitungLuas` dan `hitungKeliling`. Rumus untuk menghitung luas adalah $3.14 * \text{jari} * \text{jari}$, sedangkan untuk menghitung keliling menggunakan rumus $3.14 * 2 * \text{jari}$.

Pembahasan Lingkaran

Untuk membuat class `Lingkaran`, perlu dipikirkan apa saja atribut yang dimiliki oleh lingkaran. Ya yang dimiliki oleh lingkaran adalah jari-jari. Kita akan membuat variabel class bernama `jari2`. Method yang dimiliki lingkaran adalah `hitungLuas(self)` yang akan menghitung luas dengan rumus sesuai dengan soal, demikian juga dengan `hitungKeliling(self)` yang akan menghitung keliling sesuai rumus. Kedua method tersebut mengembalikan nilai hasil perhitungan dengan keyword `return`. Kode lengkapnya dalam Python adalah sebagai berikut:

```

1 import math
2 class Lingkaran:
3     _jari2 = 0

```

```

4
5     def __init__(self,jari2):
6         self._jari2 = jari2
7
8     def hitungLuas(self):
9         return math.pi * self._jari2 * self._jari2
10
11    def hitungKeliling(self):
12        return math.pi * 2 * self._jari2

```

program di atas disimpan dalam file bernama lingkaran.py.
Sedangkan program utama untuk menggunakan class Lingkaran adalah:

```

1  from lingkaran import Lingkaran
2
3  l = Lingkaran(7)
4  print("Luas : " + str(l.hitungLuas()))
5  print("Keliling : " + str(l.hitungKeliling()))

```

program di atas disimpan dalam file bernama main-lingkaran.py

Titik

Buatlah class Titik yang menggambarkan suatu titik koordinat di sumbu kartesian, yang memiliki atribut x dan y. Method titik tersebut adalah: hitung jarak dengan titik lain sebagai parameter, titik dicerminkan ke sumbu x, dan titik dicerminkan ke sumbu y.

Pembahasan Titik

Class Titik adalah sebuah koordinat di sumbu kartesian x dan y. Sehingga perlu dua atribut x dan y. Kemudian method yang dibuat adalah menghitung jarak antara titik sekarang dengan suatu Titik lain (sebagai parameter di dalam method tersebut). Untuk menghitung jarak digunakan rumus Euclidean distance seperti yang sudah diajarkan sebelumnya atau bisa dicari di internet sebagai berikut:

$$jarak = \sqrt{[(x_i - x_j)^2 + (y_i - y_j)^2]}$$

Gambar 5.1: Rumus Jarak Euclidian Distance

Demikian juga dengan method cerminX dan cerminY rumusnya bisa dicari di internet sebagai berikut:

Cermin terhadap sumbu X = A' (x , -y)

Cermin terhadap sumbu Y = A' (-x , y)

Program dalam Python adalah sebagai berikut:

```

1  import math
2  class Titik:
3      _x = 0
4      _y = 0
5

```

```
6     def __init__(self,x,y):
7         self._x = x
8         self._y = y
9
10    def getX(self):
11        return self._x
12
13    def getY(self):
14        return self._y
15
16    def jarak(self, t):
17        return math.sqrt(math.pow(self._x - t.getX(),2) + math.pow(self._y - t.getY(),2))
18
19    def cerminX(self):
20        return self._x,-1*self._y
21
22    def cerminY(self):
23        return -1*self._x,self._y
```

Pada program di atas harus digunakan import math karena harus menggunakan fungsi sqrt dan pow. Sedangkan untuk method harus dibuat method getX dan getY untuk mengambil nilai x dan y dari class Lingkaran yang bersifat private. Program ini disimpan dalam file bernama titik.py.

Sedangkan cara penggunaanya adalah sebagai berikut:

```
1 from titik import Titik
2 t = Titik(3,5)
3 t2 = Titik(4,8)
4
5 print(t.jarak(t2)) #3.1622776601683795
6 print(t.cerminX()) #(3, -5)
7 print(t.cerminY()) #(-3,5)
```

5.5 Unguided

Bagian unguided akan diberikan secara terpisah pada saat praktikum di Lab.

Part 02: Linear Data Structures

6	Linked List	65
6.1	Tujuan Praktikum	
6.2	Alat dan Bahan	
6.3	Materi dan Latihan Guided	
6.4	Latihan Unguided	
7	Stack	75
7.1	Tujuan Praktikum	
7.2	Alat dan Bahan	
7.3	Materi dan Latihan Guided	
7.4	Latihan Unguided	
8	Queue dan Deque	81
8.1	Tujuan Praktikum	
8.2	Alat dan Bahan	
8.3	Materi dan Latihan Guided	
8.4	Guided	
8.5	Latihan Unguided	
9	Priority Queue	89
9.1	Tujuan Praktikum	
9.2	Alat dan Bahan	
9.3	Materi	
9.4	Guided	
9.5	Unguided	

6.3 Materi dan Latihan Guided

6.3.1 Linked List

Linked list merupakan salah satu jenis struktur data linier yang memiliki ciri: penelusuran (traversing) datanya hanya bisa bersifat sekuensial (terurut) dari awal sampai akhir, atau sebaliknya dari akhir menuju ke awal. Traversingnya tidak bisa dilakukan secara acak karena struktur data linked list merupakan struktur data yang tempat penyimpanannya terhubung satu sama lain di dalam memory dengan menggunakan pointer (tanda penghubung), sehingga tidak boleh terputus. Ilustrasinya mirip dengan tempat penyimpanan data di harddisk yang bersifat sekuensial juga. Linked list bersifat dinamis (dapat ditambah dan dihapus secara dinamis) di dalam memory komputer.

Perbedaan linked list tampak jelas jika dibandingkan dengan struktur data array (list). Struktur data array (list) merupakan struktur data yang tidak harus diakses terurut, bersifat statis di dalam memory komputer. Ilustrasi perbedaannya dapat dilihat pada gambar 6.1.

	Lookup	Append (Push)	Prepend (Unshift)	Insert	Delete
Arrays	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Linked Lists	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$

Gambar 6.1: Perbedaan array dan linked list

Linked list dapat dibedakan menjadi beberapa dua jenis, yaitu: single linked list dan double linked list. Namun secara arah pointer, juga dapat dibagi lagi menjadi circular linked list atau non circular linked list. Sedangkan menurut urutannya, dapat dibagi lagi menjadi unordered linked list dan ordered linked list.

Dalam menelusuri linked list, dibutuhkan pointer penanda yang dapat bergerak dari satu node ke node lainnya. Pointer penanda tersebut biasanya dibedakan menjadi dua, yaitu penanda head dan penanda tail. Head adalah penanda awal (kepala) dan tail adalah penanda akhir (ekor). Dalam pembuatannya kebanyakan menggunakan 2 buah penanda sekaligus agar lebih mudah pembuatan programnya.

Ilustrasi linked list secara umum dapat dilihat pada gambar

Dari ilustrasi di atas, sebuah linked list berupa kumpulan node-node yang terdiri dari bagian data dan bagian pointer. Node-node tersebut terhubung satu sama lain dengan menggunakan pointer.

6.3.2 Single Linked List

Dalam pembagiannya linked list dapat dibagi menjadi 2 bagian yaitu single linked list dan double linked list. Single linked list merupakan linked list yang hanya memiliki 1 pointer penghubung saja, dan arahnya ke kanan dari satu node ke node lainnya. Single linked list memiliki kelemahan dalam hal membaca data dari belakang karena pointernya selalu maju (ke kanan) saja.

Dalam pemrograman Python untuk membuat single linked list (SLL) maka langkah pertama adalah membuat class Node terlebih dahulu. Langkah berikutnya adalah membuat class SLL. Pada contoh pertama kita akan membuat SLL yang bersifat non circular terlebih dahulu. Single Linked List non Circular (SLLNC) merupakan linked list yang hanya memiliki 1 pointer saja dan ketika data sudah berada di akhir, pointer terakhir tidak berputar lagi ke data pertamanya, sehingga disebut tidak sirkular (non circular). Ilustrasi SLLNC dapat dilihat pada gambar 6.2 sebelumnya.

Class Node dapat dibuat dengan kode sebagai berikut:

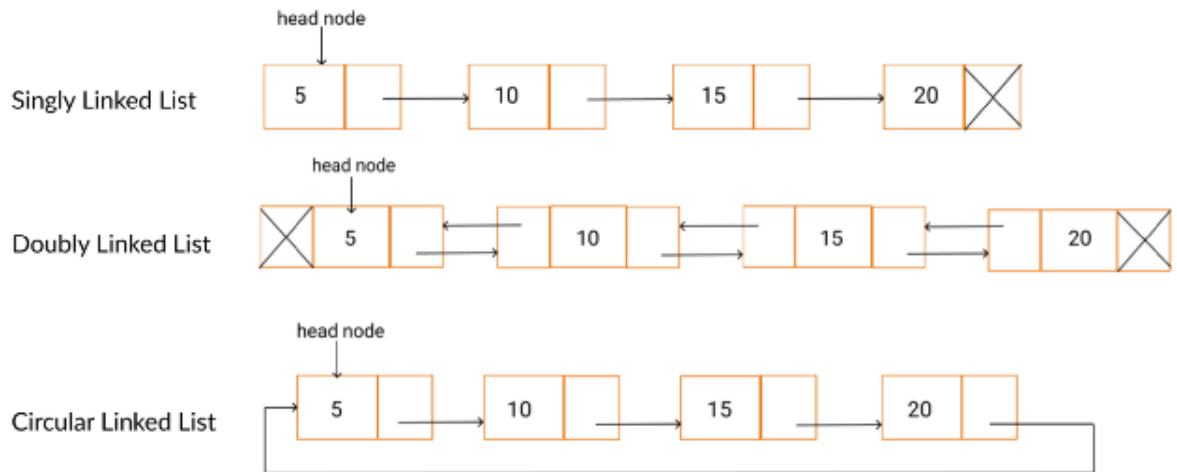
```

1 class Node:
2     def __init__(self, element, n):
3         self._element = element
4         self._next = n

```



Types of Linked List



Gambar 6.2: Ilustrasi Linked List

Dari kode di atas, dapat dilihat bahwa class Node hanya memiliki 1 konstruktor dan memiliki 2 atribut element (datanya) dan 1 pointer bernama next yang menunjuk ke node yang serupa dengannya di kanannya.

Sedangkan class SLLNC akan dibuat dengan 1 penanda kepala (head) dan 1 penanda ekor (tail) sebagai berikut:

```

1 class SLLNC:
2     def __init__(self):
3         self._head=None
4         self._tail=None
5         self._size = 0
6
7     def __len__(self):
8         return self._size
9
10    def isEmpty(self):
11        return self._size == 0

```

Method len digunakan untuk mengetahui panjang / jumlah data, sedangkan method isEmpty digunakan untuk memeriksa apakah linked list kosong atau sudah ada datanya.

Fitur SLLNC yang harus dibuat adalah sesuai dengan kemampuan dasar struktur data yaitu: tambah data (di depan dan di belakang), hapus data (dari depan dan dari belakang), dan tampil data. Sebenarnya masih bisa ditambahkan cari data, sisip data, urutkan data dan lain-lain tergantung kebutuhan.

Method tambah data dari depan adalah sebagai berikut:

```

1 def addElementHead(self,e):

```

```

2     baru = Node(e, None)
3     if self.isEmpty()==True:
4         self._head = baru
5         self._tail = baru
6         self._tail._next = None
7     else:
8         baru._next = self._head
9         self._head = baru
10    self._size += 1
11    print("Data masuk head!")

```

Untuk menambah data dari depan, pertama diperiksa terlebih dahulu apakah list kosong atau tidak, jika kosong maka ditambahkan data pertama terlebih dulu dimana semua penanda head dan tail akan menunjuk data baru yang akan masuk. Selain itu penambahan dilakukan di depan sehingga data baru akan menunjuk ke node head dan kemudian head akan bergeser ke data pertama. Ingat head harus selalu di data pertama (terdepan).

Method tambah data dari belakang adalah sebagai berikut:

```

1 def addElementTail(self,e):
2     baru = Node(e, None)
3     if self._tail == None:
4         self._head = baru
5         self._tail = baru
6         self._tail._next = None
7     else:
8         self._tail._next = baru
9         self._tail = baru
10    self._size += 1
11    print("Data masuk tail!")

```

Untuk menambah data dari belakang, pertama diperiksa terlebih dahulu apakah list kosong atau tidak, jika kosong maka ditambahkan data pertama terlebih dulu dimana semua penanda head dan tail akan menunjuk data baru yang akan masuk. Selain itu penambahan dilakukan di belakang sehingga data baru akan ditunjuk oleh node tail dan kemudian tail akan bergeser ke data terakhir. Ingat tail harus selalu di data terakhir (terbelakang).

Method hapus data dari depan adalah sebagai berikut:

```

1 def deleteFirst(self):
2     if self.isEmpty()==False:
3         d = ""
4         if self._head._next==None:
5             d = self._head._element
6             self._head=None
7             self._tail=None
8         else:
9             hapus = self._head
10            d = hapus._element
11            self._head = self._head._next
12            hapus._next=None

```

```

13         del hapus
14         self._size -= 1
15         print(d, " terhapus!")
16     else:
17         print("Kosong!")

```

Untuk menghapus data dari depan, pertama diperiksa terlebih dahulu apakah list tinggal satu data atau tidak, jika tinggal satu maka data satu-satunya tersebut akan dihapus dengan cara head dan tail menunjuk ke None (null), sedangkan jika masih ada data lain maka head sebagai data yang akan dihapus geser ke node setelahnya. Kemudian data pertama dihapus dengan perintah del. Ingat head tidak boleh dihapus saat masih ada datanya sehingga harus geser terlebih dahulu.

Method untuk hapus dari belakang adalah sebagai berikut:

```

1 def deletelast(self):
2     if self.isEmpty() == False:
3         d = None
4         bantu = self._head
5         if(self._head != self._tail):
6             while bantu._next != self._tail:
7                 bantu = bantu._next
8             hapus = self._tail
9             self._tail = bantu
10            d = hapus._element
11            del hapus
12            self._tail._next = None
13        else:
14            d = self._tail._element
15            self._head=self._tail=None
16            self._size -= 1
17            print(d, " terhapus!")
18    else:
19        print("Kosong!")

```

Untuk menghapus data dari depan, pertama diperiksa terlebih dahulu apakah list tinggal satu data atau tidak, jika tinggal satu maka data satu-satunya tersebut akan dihapus dengan cara head dan tail menunjuk ke None (null), sedangkan jika masih ada data lain, maka akan dilakukan dengan cara yang berbeda. Karena yang dihapus adalah data terakhir maka node sebelum tail harus ditunjuk oleh penanda bantu agar nantinya bantu akan ditunjuk oleh tail, sedangkan tail yang lama akan ditunjuk oleh penanda hps. Tail akan berpindah ke node sebelumnya yang ditandai dengan penanda bantu, kemudian penanda hps dihapus dengan perintah del. Ingat penanda tail tidak boleh dihapus selama masih ada data.

Method untuk menampilkan data:

```

1 def printAll(self):
2     if self.isEmpty()==False:
3         bantu = self._head
4         while(bantu!=None):
5             print(bantu._element, " ",end='')
6             bantu = bantu._next

```

```

7         print()
8     else:
9         print("Kosong")

```

Untuk menampilkan data dilakukan dengan menggunakan penanda bantu karena kita akan bergerak dari head hingga tail dan head dan tail tidak boleh berubah / bergerak posisinya. Bantu akan bergerak dari head hingga tail dengan perintah looping misalnya dengan while. Setiap kali bergerak bantu akan digunakan untuk menampilkan elemen pada node yang dikunjungi hingga akhirnya semua data ditampilkan.

Sedangkan program utamanya adalah sebagai berikut:

```

1  mydllnc = DLLNC()
2  mydllnc.addElementHead("3")
3  mydllnc.addElementHead("2")
4  mydllnc.addElementHead("1")
5  mydllnc.addElementTail("4")
6  mydllnc.addElementTail("5")
7  mydllnc.printAll()
8  mydllnc.deleteFirst()
9  mydllnc.printAll()
10 mydllnc.deleteLast()
11 mydllnc.printAll()

```

6.3.3 Double Linked List

Double linked list merupakan linked list yang hanya memiliki 2 pointer penghubung, yang arahnya ke kiri (node sebelumnya) dan ke kanan (node selanjutnya). Double linked list memiliki kelebihan dalam hal membaca data dari belakang karena pointer-nya bisa maju dan mundur, ke node setelah dan node sebelumnya.

Dalam pemrograman Python untuk membuat double linked list (DLL) maka langkah pertama adalah membuat class Node terlebih dahulu. Langkah berikutnya adalah membuat class DLL. Pada contoh pertama kita akan membuat DLL yang bersifat non circular terlebih dahulu. Double Linked List non Circular (DLLNC) merupakan linked list yang memiliki 2 pointer dan ketika data sudah berada di akhir, pointer terakhir tidak berputar lagi ke data pertamanya, sehingga disebut tidak sirkular (non circular). Ilustrasi DLLNC dapat dilihat pada gambar 6.2 sebelumnya.

Class Node dapat dibuat dengan kode sebagai berikut:

```

1 class Node:
2     def __init__(self, element, n, p):
3         self._element = element
4         self._next = n
5         self._prev = p

```

Dari kode di atas, dapat dilihat bahwa class Node hanya memiliki 1 konstruktor dan memiliki 3 atribut element (datanya) dan 1 pointer bernama next yang menunjuk ke node yang serupa dengannya di kanannya, serta 1 pointer bernama prev untuk menunjuk node serupa di kirinya.

Sedangkan class DLLNC akan dibuat dengan 1 penanda kepala (head) dan 1 penanda ekor (tail) sebagai berikut:

```

1 class DLLNC:
2     def __init__(self):
3         self._head=None
4         self._tail=None
5         self._size = 0
6
7     def __len__(self):
8         return self._size
9
10    def isEmpty(self):
11        return self._size == 0

```

Method len digunakan untuk mengetahui panjang / jumlah data, sedangkan method isEmpty digunakan untuk memeriksa apakah linked list kosong atau sudah ada datanya.

Fitur DLLNC yang harus dibuat adalah sesuai dengan kemampuan dasar struktur data yaitu: tambah data (di depan dan di belakang), hapus data (dari depan dan dari belakang), dan tampil data. Sebenarnya masih bisa ditambahkan cari data, sisip data, urutkan data dan lain-lain tergantung kebutuhan.

Method tambah data dari depan adalah sebagai berikut:

```

1 def addElementHead(self,e):
2     baru = Node(e, None, None)
3     if self.isEmpty()==True:
4         self._head = baru
5         self._tail = baru
6         self._head._next = None
7         self._head._prev = None
8         self._tail._next = None
9         self._tail._prev = None
10    else:
11        baru._next = self._head
12        self._head._prev = baru
13        self._head = baru
14        self._size += 1
15    print("Data masuk head!")

```

Untuk menambah data dari depan, pertama diperiksa terlebih dahulu apakah list kosong atau tidak, jika kosong maka ditambahkan data pertama terlebih dulu dimana semua penanda head dan tail akan menunjuk data baru yang akan masuk. Selain itu penambahan dilakukan di depan sehingga data baru akan menunjuk ke node head dan kemudian head akan bergeser ke data pertama. Ingat head harus selalu di data pertama (terdepan).

Method tambah data dari belakang adalah sebagai berikut:

```

1 def addElementTail(self,e):
2     baru = Node(e, None, None)
3     if self._tail == None:
4         self._head = baru
5         self._tail = baru

```

```

6         self._head._next = None
7         self._head._prev = None
8         self._tail._next = None
9         self._tail._prev = None
10        else:
11            self._tail._next = baru
12            baru._prev = self._tail
13            self._tail = baru
14            self._tail._next = None
15            self._size += 1
16        print("Data masuk tail!")

```

Untuk menambah data dari belakang, pertama diperiksa terlebih dahulu apakah list kosong atau tidak, jika kosong maka ditambahkan data pertama terlebih dulu dimana semua penanda head dan tail akan menunjuk data baru yang akan masuk. Selain itu penambahan dilakukan di belakang sehingga data baru akan ditunjuk oleh node tail dan kemudian tail akan bergeser ke data terakhir. Ingat tail harus selalu di data terakhir (terbelakang).

Method hapus data dari depan adalah sebagai berikut:

```

1  def deleteFirst(self):
2      if self.isEmpty()==False:
3          d = ""
4          if self._head._next==None:
5              d = self._head._element
6              self._head=None
7              self._tail=None
8          else:
9              hapus = self._head
10             d = hapus._element
11             self._head = self._head._next
12             self._head._prev = None
13             del hapus
14             self._size -= 1
15             print(d, " terhapus!")
16         else:
17             print("Kosong!")

```

Untuk menghapus data dari depan, pertama diperiksa terlebih dahulu apakah list tinggal satu data atau tidak, jika tinggal satu maka data satu-satunya tersebut akan dihapus dengan cara head dan tail menunjuk ke None (null), sedangkan jika masih ada data lain maka head sebagai data yang akan dihapus geser ke node setelahnya. Kemudian data pertama dihapus dengan perintah del. Ingat head tidak boleh dihapus saat masih ada datanya sehingga harus geser terlebih dahulu.

Method untuk hapus dari belakang adalah sebagai berikut:

```

1  def deleteLast(self):
2      if self.isEmpty() == False:
3          d = None
4          bantu = self._head
5          if(self._head._next != None):

```

```

6         hapus = self._tail
7         self._tail = self._tail._prev
8         d = hapus._element
9         self._tail._next = None
10        del hapus
11    else:
12        d = self._tail._element
13        self._head=self._tail=None
14        self._size -= 1
15        print(d, " terhapus!")
16    else:
17        print("Kosong!")

```

Untuk menghapus data dari depan, pertama diperiksa terlebih dahulu apakah list tinggal satu data atau tidak, jika tinggal satu maka data satu-satunya tersebut akan dihapus dengan cara head dan tail menunjuk ke None (null), sedangkan jika masih ada data lain, maka akan dilakukan dengan cara yang berbeda. Karena yang dihapus adalah data terakhir maka node sebelum tail harus ditunjuk oleh penanda bantu agar nantinya bantu akan ditunjuk oleh tail, sedangkan tail yang lama akan ditunjuk oleh penanda hps. Tail akan berpindah ke node sebelumnya yang ditandai dengan penanda bantu, kemudian penanda hps dihapus dengan perintah del. Ingat penanda tail tidak boleh dihapus selama masih ada data.

Method untuk menampilkan data:

```

1 def printAll(self):
2     if self.isEmpty()==False:
3         bantu = self._head
4         while(bantu!=self._tail._next):
5             print(bantu._element, " ",end='')
6             bantu = bantu._next
7         print()
8     else:
9         print("Kosong")

```

Untuk menampilkan data dilakukan dengan menggunakan penanda bantu karena kita akan bergerak dari head hingga tail dan head dan tail tidak boleh berubah / bergerak posisinya. Bantu akan bergerak dari head hingga tail dengan perintah looping misalnya dengan while. Setiap kali bergerak bantu akan digunakan untuk menampilkan elemen pada node yang dikunjunginya hingga akhirnya semua data ditampilkan.

Sedangkan program utamanya adalah sebagai berikut:

```

1 mydllnc = DLLNC()
2 mydllnc.addElementHead("3")
3 mydllnc.addElementHead("2")
4 mydllnc.addElementHead("1")
5 mydllnc.addElementTail("4")
6 mydllnc.addElementTail("5")
7 mydllnc.printAll()
8 mydllnc.deleteFirst()
9 mydllnc.printAll()

```

```

10 mydllnc.deleteLast()
11 mydllnc.printAll()

```

6.3.4 Single Linked List dan Double Linked List Circular

Setelah single linked list dan double linked list non circular dipelajari, pada bagian ini akan dijelaskan konsep untuk jenis linked list yang bersifat circular. Circular berarti penunjuk arah (pointer) antar node pada akhir data akan menunjuk ke node terdepan kembali sehingga pointernya berputar dan tidak ada yang menunjuk ke None / null.

Akibat dari bentuk circular tersebut terdapat beberapa sifat yang menarik yaitu: kemudahan linked list circular untuk membaca data dari belakang, namun untuk menampilkan data dari satu node sampai node akhir lebih sulit karena node terakhir lebih sulit ditemukan.

Bentuk linked list circular dapat dilihat dari bentuk node satuannya yang dapat dilihat dari class Node sebagai berikut: Node Single Linked List Circular:

```

1 class Node:
2     def __init__(self, element, n):
3         self._element = element
4         self._next = self

```

Node Double Linked List Circular:

```

1 class Node:
2     def __init__(self, element, n):
3         self._element = element
4         self._next = self
5         self._prev = self

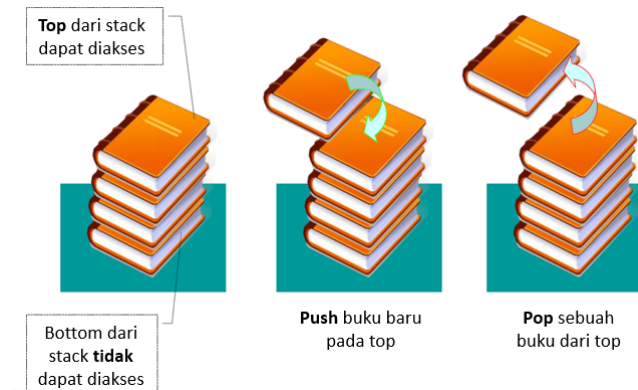
```

Class berikutnya yang harus dibuat adalah class SLLC dan DLLC dengan method seperti yang sudah dibuat pada contoh-contoh sebelumnya.

6.4 Latihan Unguided

Bagian unguided akan diberikan secara terpisah pada saat praktikum di Lab.

tumpukan, sehingga bisa memiliki status penuh. Ilustrasi stack dapat dilihat pada gambar 7.1 Pada



Gambar 7.1: Ilustrasi Stack

pemrograman Python, stack dapat diimplementasikan secara statis dengan array (list) ataupun secara dinamis menggunakan linked list seperti yang sudah dipelajari sebelumnya.

Method-method umum pada Stack adalah:

1. `len` : digunakan untuk menampilkan banyak data
2. `isEmpty` : digunakan untuk mengetahui apakah stack kosong atau tidak
3. `push` : digunakan untuk mengisi / menambahkan data baru ke stack dari salah satu sisi (dari atas tumpukan)
4. `pop` : digunakan untuk menghapus data teratas dari stack
5. `top` : digunakan untuk menampilkan data teratas (read only)
6. `print` : digunakan untuk menampilkan semua data dari teratas sampai terbawah

7.3.2 Stack dengan List

List merupakan jenis struktur data yang bersifat statis. Pemesanan tempat pada list di memory diakses dengan menggunakan indeksnya. List dibuat pada bagian method init dengan data kosong []. Method untuk mengetahui stack kosong adalah dengan memanfaatkan method `len` dari list. Method `push` menggunakan method `append` dari list untuk menambah data dari belakang (end of stack). Berarti sisi yang digunakan hanya sisi belakang saja (end of stack). Method `pop` menggunakan method `pop` dari list, sedangkan method `printAll` menggunakan cara bebas, bisa berupa looping satu persatu atau method `join string` jika datanya berupa string.

Untuk kode program class `ArrayStack` dapat dilihat dan dicoba seperti kode di bawah ini.

```

1 class ArrayStack:
2     def __init__(self):
3         self._data = []
4
5     def __len__(self):
6         return len(self._data)
7
8     def is_empty(self):
9         return len(self._data) == 0
10
11    def push(self, e):
12        self._data.append(e)
13

```

```

14     def top(self):
15         if self.is_empty():
16             print("Empty!")
17         return self._data[-1]
18
19     def pop(self):
20         if self.is_empty():
21             print("Empty!")
22         return self._data.pop()
23
24     def printAll(self):
25         if self.is_empty():
26             print("Empty!")
27         else:
28             return " ".join(self._data)

```

Cara penggunaan:

```

1 stack = ArrayStack()
2 stack.push("5")
3 stack.push("6")
4 stack.push("7")
5 print(stack.printAll())
6 stack.pop()
7 print(stack.is_empty())

```

7.3.3 Stack dengan Linked List

Linked list merupakan jenis struktur data yang bersifat dinamis menggunakan penunjuk memory. Pada contoh ini, stack dibuat dengan menggunakan single linked list yang hanya menggunakan variabel penunjuk head saja. Class Node harus dibuat dengan konsep single linked list (hanya memiliki 1 pointer next saja) dan bersifat non circular. Class implementasi diberi nama Stack yang memanfaatkan class Node yang sudah dibuat sebelumnya. Method-method yang dibuat juga mirip dengan method standard dari Stack seperti isEmpty, printAll, push, pop, dan top.

Class Node dibuat dengan kode berikut:

```

1 class Node:
2     def __init__(self, element, n):
3         self._element = element
4         self._next = n

```

Proses method push dilakukan seperti SLLNC tambah data di depan (head) dan method pop dilakukan seperti SLLNC hapus data di depan (head). Sedangkan method top dilakukan dengan menampilkan data pada head. Method printAll dilakukan dengan loop data-data dari head sampai dengan data terakhir yang ditandai dengan None.

Kode program class Stack dengan SLLNC dapat dilihat dan dicoba seperti kode di bawah ini.

```

1 class Stack: #SLLNC dengan head
2     def __init__(self):

```

```
3         self._head=None
4         self._size = 0
5
6     def __len__(self):
7         return self._size
8
9     def isEmpty(self):
10        return self._size == 0
11
12    def push(self,e):    #sama dengan tambah depan
13        baru = Node(e, None)
14        if self.isEmpty()==True:
15            self._head = baru
16            self._tail = baru
17            self._tail._next = None
18        else:
19            baru._next = self._head
20            self._head = baru
21        self._size += 1
22        #print("Data masuk ke stack!")
23
24    def top(self):    #tampilkan data di head
25        if self.isEmpty == True:
26            return "Stack kosong!"
27        else:
28            return self._head._element
29
30    def pop(self):    #hapus depan
31        if self.isEmpty()==False:
32            d = ""
33            if self._head._next==None:
34                d = self._head._element
35                self._head=None
36            else:
37                hapus = self._head
38                d = hapus._element
39                self._head = self._head._next
40                hapus._next=None
41            del hapus
42            self._size -= 1
43            #print(d, " dihapus!")
44        else:
45            print("Stack kosong!")
46        return d
47
48    def printAll(self):    #loop
49        if self.isEmpty()==False:
50            bantu = self._head
51            while(bantu!=None):
```

```
52         print(bantu._element, " ", end='')
53         bantu = bantu._next
54         print()
55     else:
56         print("Kosong")
```

Sedangkan contoh cara penggunaan class Stack adalah sebagai berikut:

```
1  #main program
2  mystack = Stack()
3  mystack.push("3")
4  mystack.push("2")
5  mystack.push("1")
6  mystack.printAll()
7  mystack.pop()
8  mystack.printAll()
9  mystack.pop()
10 mystack.printAll()
```

7.4 Latihan Unguided

Bagian unguided akan diberikan secara terpisah pada saat praktikum di Lab.

8. Queue dan Deque

8.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menjelaskan dan menggunakan struktur data linked list Queue
2. Dapat menggunakan Queue dalam studi kasus tertentu

8.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

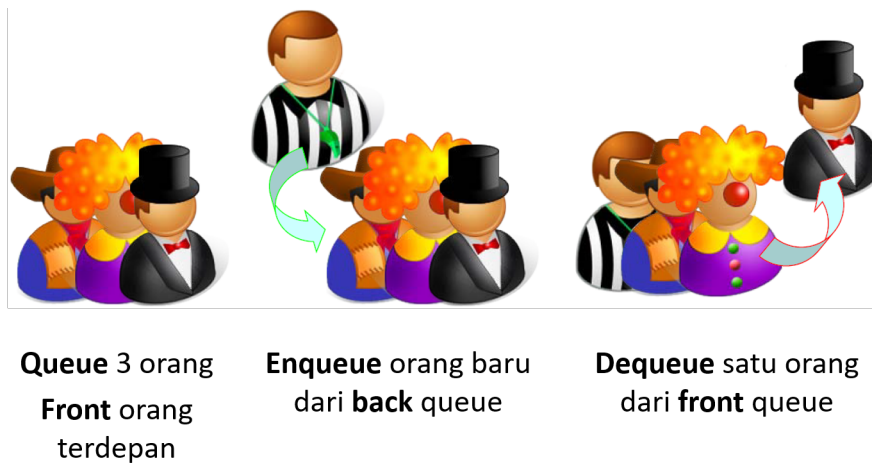
1. Python 3.8, 3.9, atau 3.10 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).
6. Moodle UKDW

8.3 Materi dan Latihan Guided

8.3.1 Queue

Queue berarti antrian. Sistem queue adalah suatu struktur data yang menggunakan konsep antrian. Antrian apapun. Di dalam antrian orang misalnya, orang akan masuk melalui bagian belakang, dan akan keluar melalui bagian depan. Jadi pintu masuk dan keluar struktur data queue

terdiri dari 2 sisi, yaitu depan dan belakang. Queue juga memiliki batas (ukuran) daya tampung antrian, sehingga bisa memiliki status penuh. Ilustrasi queue dapat dilihat pada gambar 8.1



Gambar 8.1: Ilustrasi Queue

Method-method umum pada Queue adalah:

1. `len` : digunakan untuk menampilkan banyak data
2. `isEmpty` : digunakan untuk mengetahui apakah queue kosong atau tidak
3. `enqueue` : digunakan untuk mengisi / menambahkan data baru ke queue dari sisi belakang
4. `dequeue` : digunakan untuk menghapus data dari sisi depan
5. `first` : digunakan untuk menampilkan data terdepan dari queue (read only)
6. `print` : digunakan untuk menampilkan semua data queue dari pertama sampai terakhir

Pada pemrograman Python, queue dapat diimplementasikan secara semi statis dengan array (list) circular ataupun secara dinamis menggunakan linked list seperti yang sudah dipelajari sebelumnya.

8.3.2 Queue dengan List Circular

Array list circular merupakan list biasa yang dimanipulasi indeksinya sehingga penempatan datanya bersifat berputar, misalnya list berisi 10 data, berarti indeks dari 0 - 9, dan setelah indeks ke-9 akan menuju ke indeks ke-0 lagi dan seterusnya. Atribut class ini terdiri dari data, size, dan front. Front digunakan untuk mencatat indeks terdepan dari list yang digunakan.

```

1 class ArrayQueue:
2     DEFAULT_CAPACITY = 10
3     def __init__(self):
4         self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
5         self._size = 0
6         self._front = 0
7
8     def __len__(self):
9         return self._size
10
11    def is_empty(self):
12        return self._size == 0
13
14    def first(self):    #menampilkan data dengan indeks front

```

```

15         if self.is_empty():
16             print("Empty")
17         return self._data[self._front]
18
19     def dequeue(self): #menghapus data indeks front, kemudian indeks front geser satu ke s
20         if self.is_empty():
21             print("Empty")
22         answer = self._data[self._front]
23         self._data[self._front] = None
24         self._front = (self._front + 1) % len(self._data)
25         self._size -= 1
26         return answer
27
28     def enqueue(self, e): #menambah data ke list
29         if self._size == len(self._data):
30             self._resize(2*len(self._data))
31         avail = (self._front + self._size) % len(self._data)
32         self._data[avail] = e
33         self._size += 1
34
35     def resize(self, cap): #mengubah ukuran queue pada list
36         old = self._data
37         self._data = [None] * cap
38         walk = self._front
39         for k in range(self._size):
40             self._data[k] = old[walk]
41             walk = (1 + walk) % len(old)
42         self._front = 0

```

8.3.3 Queue dengan Linked List

Linked list merupakan jenis struktur data yang bersifat dinamis menggunakan penunjuk memory. Pada contoh ini, queue dibuat dengan menggunakan single linked list yang hanya menggunakan variabel penunjuk head dan tail. Class Node harus dibuat dengan konsep single linked list (hanya memiliki 1 pointer next saja) dan bersifat non circular. Class implementasi diberi nama Queue yang memanfaatkan class Node yang sudah dibuat sebelumnya. Method-method yang dibuat juga mirip dengan method standard dari Queue seperti isEmpty, printAll, enqueue, dequeue, first dan last.

Class Node dibuat dengan kode berikut:

```

1 class Node:
2     def __init__(self, element, n):
3         self._element = element
4         self._next = n

```

Proses method enqueue dilakukan seperti SLLNC tambah data di belakang (tail) dan method dequeue dilakukan seperti SLLNC hapus data di depan (head). Sedangkan method first dilakukan dengan menampilkan data pada head. Method printAll dilakukan dengan loop data-data dari head sampai dengan data terakhir yang ditandai dengan None.

Kode program class Queue dengan SLLNC dapat dilihat dan dicoba seperti kode di bawah ini.

```
1 class Queue: #single linked list circular head dan tail
2     def __init__(self):
3         self._head=None
4         self._tail=None
5         self._size = 0
6
7     def __len__(self):
8         return self._size
9
10    def isEmpty(self):
11        return self._size == 0
12
13    def enqueue(self,e):    #tambah belakang
14        baru = Node(e, None)
15        if self.isEmpty()==True:
16            self._head = baru
17            self._tail = baru
18            self._tail._next = self._tail
19            self._head._next = self._head
20        else:
21            self._tail._next = baru
22            self._tail = baru
23            self._tail._next = self._head
24        self._size += 1
25        #print("Data masuk ke queue!")
26
27    def first(self):    #menampilkan data di head
28        if self.isEmpty == True:
29            return "Queue kosong!"
30        else:
31            return self._head._element
32
33    def last(self): #menampilkan data di tail
34        if self.isEmpty == True:
35            return "Queue kosong!"
36        else:
37            return self._tail._element
38
39    def dequeue(self): #hapus depan
40        if self.isEmpty()==False:
41            d = ""
42            if self._head._next==None:
43                d = self._head._element
44                self._head=None
45                self._tail=None
46            else:
47                hapus = self._head
48                d = hapus._element
49                self._head = self._head._next
```

```

50         self._tail._next = self._head
51         hapus._next = None
52     del hapus
53     self._size -= 1
54     #print(d, " dihapus!")
55 else:
56     print("Queue kosong!")
57     return d
58
59 def printAll(self):
60     if self.isEmpty() == False:
61         bantu = self._head
62         while(True):
63             print(bantu._element, " ", end='')
64             bantu = bantu._next
65             if(bantu == self._tail._next):
66                 break
67         print()
68     else:
69         print("Kosong")
70

```

Cara penggunaan:

```

1  #main program
2  myqueue = Queue()
3  myqueue.enqueue("1")
4  myqueue.enqueue("2")
5  myqueue.enqueue("3")
6  myqueue.printAll()
7  myqueue.dequeue()
8  myqueue.printAll()
9  myqueue.dequeue()
10 myqueue.printAll()

```

8.3.4 Deque

Deque (Double Ended Queue) adalah sebuah Queue yang memiliki 2 jalur masuk dan keluar. Di dalam Python deque diimplementasikan dalam bentuk collections. Bentuk deque bisa dibuat dalam bentuk single linked list dengan head dan tail dan menggunakan fungsi tambahDepan, hapusDepan, tambahBelakang, dan hapusBelakang.

Pada Python deque sudah bersifat built in dalam collections. Contoh:

```

1  from collections import deque
2  # pembuatan queue
3  queue = deque(['nama', 'umur', 'tgllahir'])
4  print(queue)

```

Fungsi/method yang bisa digunakan:

- `append()` : untuk menambah elemen dari belakang (tambahBelakang)
- `appendleft()` : untuk menambah elemen dari depan (tambahDepan).
- `pop()` : untuk menghapus elemen dari belakang (hapusBelakang).
- `popleft()` : untuk menghapus elemen dari depan (hapusDepan).
- `index(ele, beg, end)` : untuk mengembalikan indeks pertama dari data yang ditemukan dari posisi beg sampai dengan end.
- `insert(i, a)` : digunakan untuk menyisipkan elemen pada indeks tertentu
- `remove(e)` : untuk menghapus elemen pertama yang ditemukan sesuai nilai elemen
- `extend(iterable)` : untuk menambah elemen berupa list dari belakang (lebih dari satu data)
- `extendleft(iterable)` : untuk menambah elemen berupa list dari depan (lebih dari satu data)
- `reverse()` : untuk membalik isi deque

8.4 Guided

Membalik Queue

Buatlah program untuk membalik Queue dengan memanfaatkan Stack yang sudah pernah dibuat sebelumnya. Misal terdapat Queue: [A, B, C] maka output program adalah Queue: [C, B, A]

8.4.1 Pembahasan

- Buatlah Queue yang berisi elemen-elemen tertentu
- Buatlah fungsi untuk membalik Queue tersebut dengan bantuan Stack. Stack digunakan untuk menyimpan semua elemen Queue yang telah dikeluarkan dari Queue. Setelah itu dari Stack keluarkan elemen satu persatu dan dimasukkan kembali ke Queue yang telah dikosongkan sebelumnya.

Langkah-langkah tersebut dapat diwujudkan dalam bentuk source code berikut ini:

```

1      from queue import Queue
2      from stack import Stack
3
4      def balikQueue(Q):
5          S = Stack()
6          str = ""
7          n = len(Q)
8          for i in range(n):
9              str = Q.first()
10             S.push(str)
11             Q.dequeue()
12         while (S.isEmpty()==False):
13             Q.enqueue(S.top())
14             S.pop()
15
16         #main program
17         ipt = Queue()
18         ipt.enqueue("A")
19         ipt.enqueue("B")
20         ipt.enqueue("C")
21         print(ipt.first())
22         balikQueue(ipt)
23         print(ipt.first())

```

8.5 Latihan Unguided

Bagian unguided akan diberikan secara terpisah pada saat praktikum di Lab.

9. Priority Queue

9.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menjelaskan dan menggunakan Priority Queue
2. Dapat menggunakan Priority Queue dalam studi kasus tertentu

9.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.8, 3.9, atau 3.10 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).
6. Moodle UKDW

9.3 Materi

9.3.1 Priority Queue

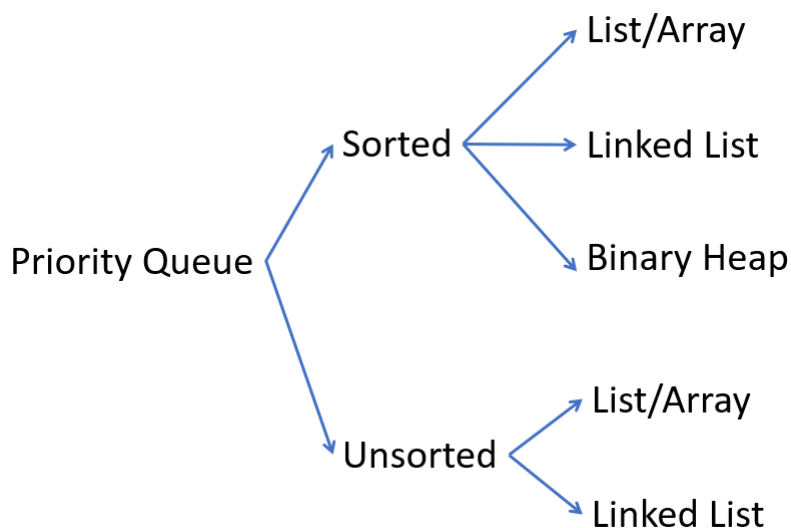
Priority Queue merupakan salah satu bentuk pengembangan dari Queue. Jika pada Queue berlaku First In First Out, maka pada Priority Queue data yang keluar ditentukan berdasarkan tingkat prioritasnya. Perbedaan utama dari Queue dan Priority Queue dapat dilihat pada Tabel 9.1.

Operasi	Queue	Priority Queue
Penambahan data	Dilakukan di belakang (append)	Bisa di belakang, bisa juga sesuai tingkat prioritas
Penghapusan data	Dilakukan pada data paling depan	Data yang memiliki tingkat prioritas paling tinggi
Pengambilan data	Data pada posisi paling depan	Data yang memiliki prioritas paling tinggi

Tabel 9.1: Perbedaan Queue dan Priority Queue

Dari Tabel 9.1 dapat disimpulkan bahwa pada Queue operasi dilakukan berdasarkan posisi data (urutan kedatangan), sedangkan pada Priority Queue operasi dilakukan berdasarkan tingkat prioritas dari data. Tingkat prioritas umumnya dalam bentuk angka dalam skala 1, 2, 3, 4, Semakin kecil nilainya, maka tingkat prioritasnya dianggap lebih tinggi.

Implementasi dari Priority Queue umumnya dibedakan menjadi dua macam, yaitu terurut (sorted) dan tidak terurut (unsorted) seperti yang ditunjukkan pada Gambar 9.1. Pada praktikum ini akan dibahas implementasi Priority Queue berbasis Linked List.



Gambar 9.1: Variasi Implementasi Priority Queue

9.3.2 Priority Queue Dengan Unsorted Linked List

Pada Unsorted Linked List, data tidak disusun secara berurutan, melainkan berdasarkan urutan kedatangan. Sebagai contoh, jika data berikut ini secara berurutan dimasukkan ke dalam Priority Queue (unsorted):

1. Data: Bambang, Prioritas: 4.
2. Data: Badu, Prioritas: 6.
3. Data: Upin, Prioritas: 2.
4. Data: Susi, Prioritas: 1.
5. Data: Mail, Prioritas: 3.
6. Data: Rose, Prioritas: 5.

Maka Priority Queue yang terbentuk dapat dilihat pada Gambar 9.2.

Front	4	6	2	1	3	5
	Bambang	Badu	Upin	Susi	Mail	Rose

Gambar 9.2: Priority Queue dengan Unsorted Linked List

Dari hasil tersebut, dapat dilihat jika operasi penambahan data pada Priority Queue (unsorted) sama persis dengan penambahan data pada Queue. Data yang paling baru dimasukkan pada posisi paling belakang, tanpa melihat tingkat prioritasnya. Kompleksitas waktu dari operasi penambahan data ini adalah $O(1)$. Kompleksitas waktu konstan artinya langkah yang diperlukan untuk memasukkan data baru selalu sama, tidak bergantung pada ukuran dari Priority Queue.

Operasi berikutnya adalah penghapusan data. Penghapusan data dilakukan berdasarkan tingkat prioritas. Data yang dihapus adalah data dengan tingkat prioritas tertinggi. Ilustrasi dari penghapusan data pada Priority Queue (unsorted) dapat dilihat pada Gambar 9.3.

Front	4	6	2	1	3	5
	Bambang	Badu	Upin	Susi	Mail	Rose

Data dengan prioritas tertinggi

↓

Front	4	6	2	3	5
	Bambang	Badu	Upin	Mail	Rose

Gambar 9.3: Penghapusan Data Pada Priority Queue (unsorted)

Operasi penghapusan data diawali dengan pencarian data dengan tingkat prioritas tertinggi terlebih dahulu. Karena itu kompleksitas waktunya adalah $O(n)$. Artinya semakin besar ukuran Priority Queue, maka waktu yang diperlukan akan semakin bertambah secara linier.

Operasi pengambilan data (Peek/Front/Min) hampir mirip dengan operasi penghapusan data. Operasi ini hanya mengambil nilai data dengan prioritas tertinggi. Untuk mendapatkan data dengan prioritas tertinggi diperlukan operasi pencarian terlebih dahulu.

Dari ketiga operasi utama tersebut, maka dapat disimpulkan kompleksitas waktu dari sebuah Priority Queue (unsorted) adalah sebagai berikut:

- Penambahan data: $O(1)$. Konstan.
- Penghapusan data: $O(n)$. Linier.
- Pengambilan data: $O(n)$. Linier.

9.3.3 Priority Queue Dengan Sorted Linked List

Pada Sorted Linked List, posisi data di dalam antrian diatur sedemikian rupa sehingga selalu teratur berdasarkan tingkat prioritasnya. Sebagai contoh, jika data berikut dimasukkan secara berurutan ke dalam Priority Queue (sorted):

1. Data: 14, Prioritas: 7.

2. Data: 9, Prioritas: 4.
3. Data: 18, Prioritas: 1.
4. Data: 25, Prioritas: 7.
5. Data: 27, Prioritas: 2.

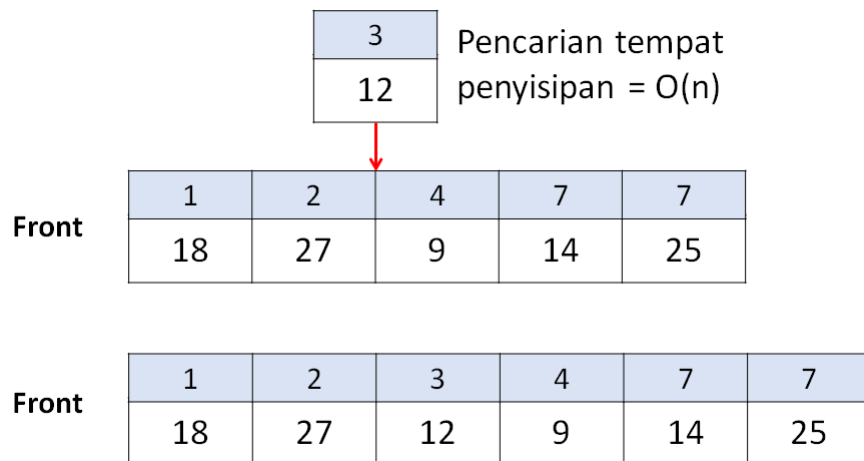
Maka Priority Queue yang terbentuk dapat dilihat pada Gambar 9.4.

1	2	4	7	7	Prioritas
18	27	9	14	25	Data

Front

Gambar 9.4: Priority Queue Dengan Sorted Linked List

Penambahan data baru diawali dengan menentukan posisi data tersebut berdasarkan tingkat prioritasnya. Operasi penambahan data harus selalu menghasilkan Priority Queue yang isinya terurut berdasarkan tingkat prioritasnya. Jika ditambahkan data baru: 12 dengan prioritas: 3, maka data tersebut akan diletakkan pada posisi seperti yang ditunjukkan oleh Gambar 9.5.



Gambar 9.5: Menambah Data Pada Priority Queue (sorted)

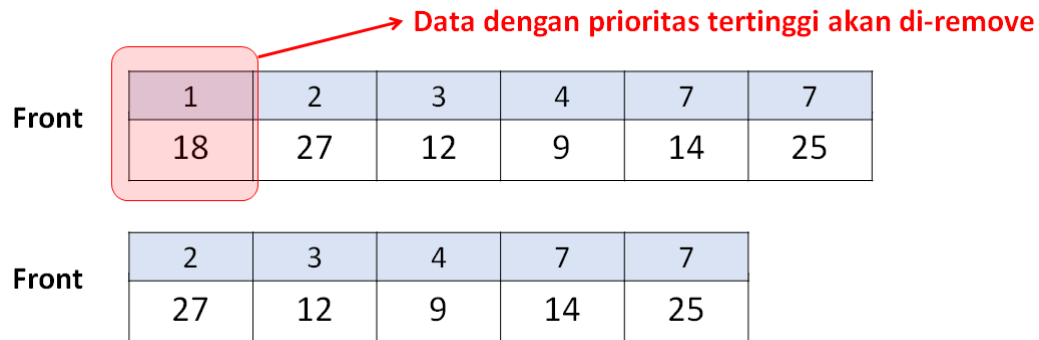
Operasi penambahan data memiliki kompleksitas waktu $O(n)$, karena harus diawali dengan menentukan posisi data yang baru berdasarkan tingkat prioritasnya. Sehingga jika ukuran Priority Queue semakin besar, maka waktu yang diperlukan akan bertambah secara linier sesuai dengan ukuran Priority Queue tersebut.

Operasi penghapusan data dilakukan pada data yang paling depan, karena bisa dipastikan memiliki tingkat prioritas yang paling tinggi. Ilustrasi dari operasi penghapusan data dapat dilihat pada Gambar 9.6.

Operasi penghapusan data memiliki kompleksitas waktu $O(1)$, dikarenakan data dengan prioritas tertinggi dipastikan posisinya sudah paling depan. Jika ukuran Priority Queue semakin besar, maka langkah yang diperlukan akan tetap sama (konstan).

Operasi pengambilan data dengan prioritas tertinggi juga memiliki kompleksitas waktu $O(1)$, sama seperti pada operasi penghapusan data. Hal ini disebabkan karena data dengan prioritas tertinggi dipastikan berada dalam posisi paling depan.

Dari ketiga operasi utama tersebut, maka dapat disimpulkan kompleksitas waktu dari sebuah Priority Queue (sorted) adalah sebagai berikut:



Gambar 9.6: Penghapusan Data Pada Priority Queue (sorted)

- Penambahan data: $O(n)$. Linier.
- Penghapusan data: $O(1)$. Konstan.
- Pengambilan data: $O(1)$. Konstan.

9.4 Guided

9.4.1 Implementasi Priority Queue (unsorted)

Pada bagian ini akan dibahas mengenai implementasi Priority Queue dengan menggunakan Unsorted Single Linked List Non Circular.

Mendefinisikan Node

Node pada linked list yang akan dibuat harus memuat informasi prioritas (dengan nilai 1, 2, 3, 4, 5, ...) dimana prioritas = 1 dianggap sebagai prioritas tertinggi. Kode program untuk mendefinisikan Node tersebut adalah sebagai berikut:

```

1
2 class Node:
3     def __init__(self, data, priority):
4         self._data = data
5         self._priority = priority          # 1 (tertinggi), 2, 3, 4, ...
6         self._next = None
7

```

Jika ingin mendefinisikan Node baru dengan data = 'Bambang' dan prioritas = 2, maka kode programnya seperti berikut ini:

```

1
2 class Node:
3     def __init__(self, data, priority):
4         self._data = data
5         self._priority = priority          # 1 (tertinggi), 2, 3, 4, ...
6         self._next = None
7
8 baru = Node('Bambang', 2)
9

```

Struktur Dasar Class PriorityQueueUnsorted

Setelah Node didefinisikan, class tersebut akan digunakan pada class PriorityQueueUnsorted yang diimplementasikan seperti berikut ini:

```

1 class PriorityQueueUnsorted:
2     def __init__(self):
3         self._head = None
4         self._tail = None
5         self._size = 0
6
7     def is_empty(self):
8         if self._size == 0:
9             return True
10        else:
11            return False
12
13    def __len__(self):
14        return self._size
15
16    def print_all(self):
17        if self.is_empty() == True:
18            print('Priority Queue is empty')
19        else:
20            bantu = self._head
21            while bantu != None:
22                print('(', bantu._data, ',', bantu._priority, ')', end=' ')
23                bantu = bantu._next
24        print()

```

Fungsi-fungsi yang telah diimplementasikan pada class PriorityQueueUnsorted adalah:

- **Fungsi __init__()** yang merupakan constructor. Tugasnya untuk menginisialisasi nilai dari head, tail dan jumlah data di dalam Priority Queue tersebut.
- **Fungsi is_empty()** yang digunakan untuk mengecek apakah Priority Queue kosong atau tidak. Fungsi ini akan berguna pada saat akan menghapus data.
- **Fungsi __len__()** yang digunakan untuk mendapatkan informasi berapa jumlah data yang ada di dalam Priority Queue.
- Fungsi print_all() yang digunakan untuk menampilkan isi dari Priority Queue.

Penambahan Data

Implementasi dari operasi penambahan data dapat dilihat pada kode program berikut ini:

```

1
2 def add(self, data, priority):
3     baru = Node(data, priority)
4     if self.is_empty():                # kosong
5         self._head = baru
6         self._tail = baru
7     else:                              # insert belakang

```

```

8         self._tail._next = baru
9         self._tail = baru
10        self._size = self._size + 1
11

```

Operasi penambahan data dilakukan dengan cara menambah data di bagian belakang (tail). Yang perlu diperhatikan adalah penambahan data di saat Priority Queue masih kosong, maka data yang baru akan menjadi head dan tail sekaligus.

Penghapusan Data

Operasi penghapusan data dilakukan dengan langkah-langkah berikut ini:

1. Cari posisi data dengan tingkat prioritas tertinggi.
2. Perlu pengecekan data dengan tingkat prioritas tertinggi tersebut posisinya ada di head, atau tail, atau di tengah. Hal ini perlu dilakukan karena cara menghapusnya akan berbeda beda.
3. Hapus data dengan prioritas tertinggi.

Operasi penghapusan data dapat diimplementasikan dengan cara yang ditunjukkan oleh kode program berikut ini:

```

1
2 def remove(self):                                # implementasi ini tidak return
3     if self.is_empty() == False:
4         if self._size == 1:
5             bantu = self._head
6             self._head = None
7             self._tail = None
8             del bantu
9         else:
10            # ambil prioritas pada head sebagai prioritas tertinggi yang diketahui
11            min_priority = self._head._priority
12            # cek dari head sampai tail, berapa prioritas tertinggi
13            hapus = self._head
14            while hapus != None:
15                if hapus._priority < min_priority:
16                    min_priority = hapus._priority
17                hapus = hapus._next
18            # prioritas tertinggi sudah diketahui, letakkan hapus di node tersebut
19            hapus = self._head
20            while hapus._priority != min_priority:
21                hapus = hapus._next
22            # cek yang akan dihapus itu head, tail, atau tengah?
23            if hapus == self._head:
24                # hapus head
25                self._head = self._head._next
26                del hapus
27            else:
28                # hapus tail atau tengah caranya sama saja
29                # letakkan bantu di posisi sebelum hapus
30                bantu = self._head
31                while bantu._next != hapus:

```

```

32         bantu = bantu._next
33         # hapus node
34         bantu._next = hapus._next
35         del hapus
36         # pastikan tail di posisi paling belakang
37         self._tail = self._head
38         while self._tail._next != None:
39             self._tail = self._tail._next
40         self._size = self._size - 1
41

```

Pengambilan Data

Operasi pengambilan data dengan prioritas tertinggi dilakukan dengan cara yang hampir sama seperti penghapusan, yaitu harus mencari posisi data dengan prioritas tertinggi. Implementasi dari pengambilan data ditunjukkan oleh kode program berikut ini:

```

1  def peek(self):    # return dalam bentuk tuple (data, priority)
2      if self.is_empty() == True:
3          return None
4      else:
5          if self._size == 1:
6              return tuple([self._head._data, self._head._priority])
7          else:
8              min_priority = self._head._priority
9              bantu = self._head
10             # cari nilai prioritas tertinggi
11             while bantu != None:
12                 if bantu._priority < min_priority:
13                     min_priority = bantu._priority
14                 bantu = bantu._next
15             # nilai prioritas tertinggi sudah diketahui,
16             # ambil nilai dan prioritas dari node tersebut
17             bantu = self._head
18             while bantu._priority != min_priority:
19                 bantu = bantu._next
20             return tuple([bantu._data, bantu._priority])

```

Pengujian Implementasi

Untuk mencoba implementasi Priority Queue (unsorted) tersebut, dapat dilakukan dengan cara yang ditunjukkan oleh kode program berikut:

```

1  myQueue = PriorityQueueUnsorted()
2  myQueue.add('Amber', 5)
3  myQueue.add('Diluc', 1)
4  myQueue.add('Beidou', 3)
5  myQueue.add('Kaeya', 4)
6  myQueue.print_all()
7  data, priority = myQueue.peek()

```

```

8 print(data)
9 print(priority)
10 myQueue.remove()
11 myQueue.print_all()
12 myQueue.remove()
13 myQueue.print_all()
14 myQueue.remove()
15 myQueue.print_all()

```

Output yang dihasilkan adalah seperti berikut:

```

( Amber , 5 ) ( Diluc , 1 ) ( Beidou , 3 ) ( Kaeya , 4 )
Diluc
1
( Amber , 5 ) ( Beidou , 3 ) ( Kaeya , 4 )
( Amber , 5 ) ( Kaeya , 4 )
( Amber , 5 )

```

9.4.2 Implementasi Priority Queue (sorted)

Pada bagian ini akan dibahas mengenai implementasi Priority Queue dengan menggunakan Sorted Double Linked List Non Circular.

Mendefinisikan Node

Node yang digunakan dalam linked list tersebut harus memuat informasi prioritas, yang didefinisikan sebagai class Node berikut:

```

1 class Node:
2     def __init__(self, data, priority):
3         self._data = data
4         self._priority = priority          # 1 (tertinggi), 2, 3, 4, ...
5         self._next = None
6         self._prev = None

```

Struktur Dasar Class PriorityQueueSorted

Class Node tersebut akan digunakan pada class PriorityQueueSorted yang didefinisikan sebagai berikut:

```

1 class PriorityQueueSorted:
2     def __init__(self):
3         self._head = None
4         self._tail = None
5         self._size = 0
6
7     def is_empty(self):
8         if self._size == 0:
9             return True
10        else:

```

```

11         return False
12
13     def __len__(self):
14         return self._size
15
16     def print_all(self):
17         if self.is_empty() == True:
18             print('Priority Queue is empty')
19         else:
20             bantu = self._head
21             while bantu != None:
22                 print('(', bantu._data, ',', bantu._priority, ')', end=' ')
23                 bantu = bantu._next
24             print()

```

Fungsi-fungsi yang telah diimplementasikan pada class `PriorityQueueSorted` adalah:

- **Fungsi `__init__()`** yang merupakan constructor. Tugasnya untuk menginisialisasi nilai dari head, tail dan jumlah data di dalam Priority Queue tersebut.
- **Fungsi `is_empty()`** yang digunakan untuk mengecek apakah Priority Queue kosong atau tidak. Fungsi ini akan berguna pada saat akan menghapus data.
- **Fungsi `__len__()`** yang digunakan untuk mendapatkan informasi berapa jumlah data yang ada di dalam Priority Queue.
- Fungsi `print_all()` yang digunakan untuk menampilkan isi dari Priority Queue.

Penghapusan Data

Operasi penghapusan data dilakukan terhadap data pada posisi paling depan (head) karena bisa dipastikan merupakan data dengan prioritas tertinggi. Kode program untuk penghapusan data adalah sebagai berikut:

```

1 def remove(self):                # implementasi ini tidak return
2     if self.is_empty() == False:
3         # yang akan dihapus pasti head
4         hapus = self._head
5         # jika cuma ada 1 node
6         if self._size == 1:
7             self._head = None
8         else:
9             self._head = self._head._next
10            self._head._prev = None
11            # hapus node
12            del hapus
13            self._size = self._size - 1

```

Pengambilan Data

Operasi pengambilan data pada Priority Queue (sorted) dilakukan dengan cara mengambil data pada posisi paling depan (head), seperti ditunjukkan pada kode program berikut:

```

1 def peek(self):
2     if self.is_empty() == True:

```

```

3         return None
4     else:
5         return tuple([self._head._data, self._head._priority])

```

Penambahan Data

Operasi penambahan data adalah operasi yang lebih sulit, karena harus dilakukan dalam beberapa langkah berikut ini:

- Cari posisi data baru berdasarkan tingkat prioritasnya.
- Gunakan variabel bantu dan bantu2 untuk melakukan penyisipan.
- Variabel bantu harus diletakkan pada posisi setelah data akan disisipkan.
- Variabel bantu2 harus diletakkan pada posisi sebelum data akan disisipkan.
- Lakukan penyisipan data.

Pada operasi penambahan data juga perlu dilakukan pengecekan terlebih dahulu apakah data yang baru akan berada di posisi paling depan (menjadi head yang baru), paling belakang (menjadi tail yang baru) atau di tengah. Implementasi dari penambahan data ditunjukkan oleh kode program berikut:

```

1  # add: harus memastikan linked list selalu terurut berdasarkan prioritas
2  def add(self, data, priority):
3      baru = Node(data, priority)
4      if self.is_empty():
5          self._head = baru
6          self._tail = baru
7      elif self._size == 1:
8          # isinya cuma 1, insert sebelum atau setelah head?
9          if self._head._priority > priority:
10             # insert sebelum head
11             baru._next = self._head
12             self._head._prev = baru
13             self._head = baru
14         else:
15             # insert setelah head
16             self._head._next = baru
17             baru._prev = self._head
18             self._tail = baru
19     else:
20         # cek apakah harus insert depan?
21         if self._head._priority > priority:
22             baru._next = self._head
23             self._head._prev = baru
24             self._head = baru
25         # cek apakah harus insert belakang?
26         elif self._tail._priority <= priority:
27             self._tail._next = baru
28             baru._prev = self._tail
29             self._tail = baru
30             self._tail._next = None
31         else:
32             # berarti insert di tengah

```

```

33         # letakkan bantu di posisi setelah insertion point
34         bantu = self._head
35         while bantu._priority < priority:
36             bantu = bantu._next
37         # gunakan bantu2 di posisi sebelum insertion point
38         bantu2 = bantu._prev
39         baru._next = bantu
40         bantu._prev = baru
41         bantu2._next = baru
42         baru._prev = bantu2
43     self._size = self._size + 1

```

Pengujian Implementasi

Untuk mencoba implementasi Priority Queue (sorted) tersebut, dapat dilakukan dengan cara yang ditunjukkan oleh kode program berikut:

```

1 myQueue = PriorityQueueSorted()
2 myQueue.add('Amber', 5)
3 myQueue.add('Diluc', 1)
4 myQueue.print_all()
5 myQueue.add('Beidou', 3)
6 myQueue.add('Kaeya', 4)
7 myQueue.print_all()
8 data, priority = myQueue.peek()
9 print(data)
10 print(priority)
11 myQueue.remove()
12 myQueue.print_all()
13 myQueue.remove()
14 myQueue.print_all()
15 myQueue.remove()
16 myQueue.print_all()
17 myQueue.add('Venti', 1)
18 myQueue.print_all()

```

Output yang dihasilkan adalah seperti berikut:

```

( Diluc , 1 ) ( Amber , 5 )
( Diluc , 1 ) ( Beidou , 3 ) ( Kaeya , 4 ) ( Amber , 5 )
Diluc
1
( Beidou , 3 ) ( Kaeya , 4 ) ( Amber , 5 )
( Kaeya , 4 ) ( Amber , 5 )
( Amber , 5 )
( Venti , 1 ) ( Amber , 5 )

```

9.5 Unguided

Bagian unguided akan diberikan secara terpisah pada saat praktikum di Lab.



Part 03: Sorting and Searching

10	Quick Sort dan Merge Sort	103
10.1	Tujuan Praktikum	
10.2	Alat dan Bahan	
10.3	Materi	
10.4	Unguided	
11	Hashing	111
11.1	Tujuan Praktikum	
11.2	Alat dan Bahan	
11.3	Materi dan Latihan Guided	

10. Quick Sort dan Merge Sort

10.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menjelaskan mengenai karakteristik algoritma Divide and Conquer
2. Dapat mengimplementasikan langkah-langkah pengurutan data menggunakan algoritma Quick Sort
3. Dapat mengimplementasikan langkah-langkah pengurutan data menggunakan algoritma Merge Sort

10.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.8, 3.9, atau 3.10 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).
6. Moodle UKDW

10.3 Materi

10.3.1 Algoritma Divide and Conquer

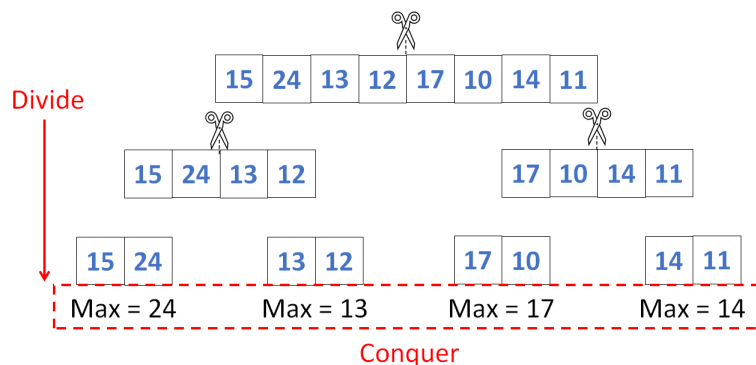
Algoritma Divide and Conquer (DaC) merupakan salah satu teknik atau paradigma yang digunakan untuk memecahkan masalah dengan cara membagi masalah menjadi masalah-masalah yang lebih kecil (sub-problem) sampai menemukan masalah terkecil yang dapat langsung diselesaikan. Seluruh masalah yang telah diselesaikan digabungkan sehingga dapat menyelesaikan masalah secara keseluruhan. Beberapa contoh algoritma yang menerapkan teknik Divide and Conquer: Quick Sort, Merge Sort, Strassen Matrix Multiplication, Longest Common Prefix, Exponential Search, Karatsuba Multiplication, Closest Pair of Points dan masih banyak algoritma-algoritma lainnya.

Algoritma DaC biasanya terdiri dari beberapa tahap sebagai berikut:

1. **Divide:** Membagi masalah menjadi masalah-masalah yang lebih kecil.
2. **Conquer:** Menyelesaikan masalah-masalah terkecil.
3. **Combine:** Menggabungkan seluruh solusi dari bagian-bagian terkecil untuk menyusun solusi yang dapat memecahkan masalah yang diberikan.

Sebagai contoh, permasalahan pencarian nilai maksimum atau minimum dari sebuah list dapat diselesaikan menggunakan teknik DaC sebagai berikut:

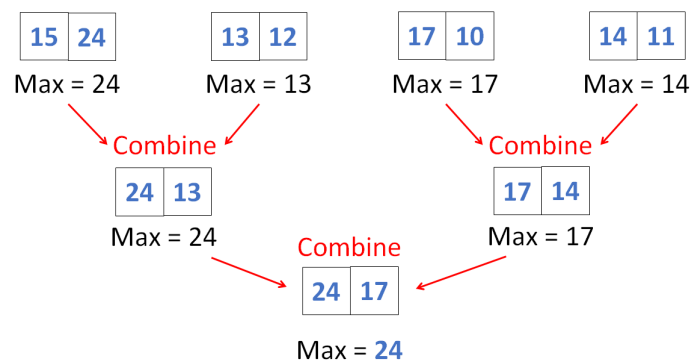
- Bagi list menjadi dua bagian sama besar terus menerus sampai potongan yang terkecil, yaitu list berisi satu atau dua elemen.
- Bandingkan nilai elemen-elemen pada potongan terkecil, tentukan nilai terbesar atau terkecilnya (sesuai masalah, maksimum atau minimum). Ilustrasi dari langkah ini dapat dilihat pada Gambar 10.1.
- Gabungkan nilai terkecil/terbesar tersebut dengan hasil dari potongan lainnya. Lakukan terus-menerus sampai didapatkan nilai terkecil/terbesar dari list secara keseluruhan. Ilustrasi dari langkah ini dapat dilihat pada Gambar 10.2.



Gambar 10.1: Tahap Divide dan Conquer untuk mencari nilai maksimum pada sebuah list

Implementasi dari DaC biasanya berupa fungsi rekursif. Untuk mencari nilai maksimal secara rekursif dengan teknik DaC, maka perlu menyusun dua bagian utama fungsi rekursif (base case dan recursive case):

- **Base case (conquer).** Pada masalah nilai maksimum, maka base case adalah pada saat potongan list hanya berisi satu atau dua elemen. Jika hanya berisi satu elemen, maka nilai maksimum adalah nilai dari elemen tersebut. Jika terdiri dari dua elemen, maka tinggal dibandingkan mana yang lebih besar nilainya dari kedua elemen tersebut.
- **Recursive case (divide dan combine).** Pembagian list menjadi dua bagian dilakukan pada bagian ini. Kemudian penggabungan hasil nilai maksimum dilakukan pada bagian ini juga,



Gambar 10.2: Tahap Combine menggabungkan seluruh nilai maksimum dan kembali dilakukan perbandingan berikutnya.

dengan cara membandingkan nilai terbesar dari potongan sebelah kiri dan sebelah kanan, untuk mendapatkan nilai maksimum keseluruhan secara bertahap.

Source code untuk fungsi maksimum dengan menggunakan teknik DaC adalah sebagai berikut:

```

1
2 # untuk nilai terkecil/terbesar
3 import math
4
5 def max(data, start, end, max_value):
6     # base case
7     if start == end:
8         max_value = data[start]
9         return max_value
10    # base case
11    elif start == end - 1:
12        if data[start] > data[end]:
13            max_value = data[start]
14            return max_value
15        else:
16            max_value = data[end]
17            return max_value
18    else:
19        # recursive case
20        mid = (start + end) // 2
21        # ruas kiri
22        left_max = max(data, start, mid, max_value)
23        # ruas kanan
24        right_max = max(data, mid+1, end, max_value)
25        # combine
26        if left_max > right_max:
27            return left_max
28        else:
29            return right_max
30

```

```

31 # bagian utama program, untuk mencoba
32 data = list([150, 24, 13, 12, 1, 10, 140, 11])
33 print(max(data, 0, len(data)-1, -math.inf))
34

```

Pada bagian recursive case dapat dilihat list dibagi menjadi dua bagian dengan mencari nilai mid. Kemudian ruas kiri dari list diproses secara rekursif terus-menerus sampai mencapai base case, yaitu potongan list yang terdiri dari satu elemen (kondisi `start == end`) dan dua elemen (kondisi `start == end - 1`). Pada kondisi base case maka nilai maksimal bisa didapatkan dengan menggunakan perbandingan sederhana. Hal yang sama juga dilakukan pada ruas kanan. Kemudian hasil nilai maksimal dari ruas kanan dan ruas kiri dibandingkan untuk mendapatkan nilai maksimal dari kedua ruas tersebut.

Teknik DaC memiliki beberapa kelebihan dan kekurangan sebagai berikut:

- (+) Mampu memecahkan masalah yang kompleks.
- (+) Menghasilkan algoritma yang efisien.
- (+) Dapat memanfaatkan paralelisme, yaitu pemrosesan secara paralel.
- (-) Karena biasanya berbentuk fungsi rekursif, maka akan lebih lambat daripada fungsi biasa. Kemudian juga ada batasan tingkat rekursif yang bisa dilakukan.
- (-) Implementasi bisa tidak efisien jika ditemukan sub-problem yang berulang-ulang, sehingga harus dikerjakan berulang-ulang.

10.3.2 Quick Sort

Quick Sort adalah algoritma untuk pengurutan data yang menerapkan teknik DaC. Secara umum berikut ini adalah langkah-langkah dari algoritma Quick sort:

- Bagi data menjadi dua partisi (partisi kiri dan kanan), yang dipisahkan oleh sebuah elemen yang dipilih menjadi pivot.
- Tukar posisi setiap elemen sehingga semua elemen di sebelah kiri pivot adalah elemen yang nilainya lebih kecil dari pivot. Demikian juga semua elemen di sebelah kanan pivot merupakan elemen yang nilainya lebih besar dari pivot.
- Setelah pivot membagi data menjadi dua partisi, maka secara rekursif lakukan kembali pembentukan pivot dari partisi kanan dan kiri sampai mencapai elemen terkecil.
- Penukaran posisi elemen dilakukan secara in-place, sehingga setelah pembagian partisi sudah mencapai partisi terkecil, maka proses pengurutan sudah selesai.

Algoritma Quick Sort merupakan algoritma pengurutan yang **tidak stabil**, sehingga elemen dengan nilai yang sama bisa saja posisinya akan berubah setelah data selesai diurutkan. Proses pengurutan dilakukan dengan menukar posisi elemen relatif terhadap pivot, yang dilakukan langsung pada data yang akan diurutkan (**in-place**). Kinerja dari Quick Sort sangat ditentukan dari pemilihan pivot, yang bisa dilakukan dengan beberapa cara berikut ini:

- Ambil elemen pertama sebagai pivot.
- Ambil elemen terakhir sebagai pivot.
- Ambil elemen pada posisi random sebagai pivot.
- Ambil elemen yang merupakan nilai tengah (median) sebagai pivot.

Algoritma Quick Sort akan mengalami kondisi **worst case** pada saat pivot yang dipilih merupakan nilai maksimum/minimum dari partisi tersebut. Kondisi terbaik (**best case**) pada saat pivot yang dipilih merupakan nilai median dari partisi tersebut.

Implementasi dari algoritma Quick Sort biasanya terdiri dari dua fungsi, yaitu fungsi `quicksort()` sebagai fungsi utama dan `partition()` yang bertugas untuk membuat partisi berdasarkan pivot. Fungsi `quicksort()` dapat dilihat pada kode program berikut ini:

```

1
2 def quicksort(data, start, end):
3     if start < end:
4         # partisi dan dapatkan posisi pivot
5         pivot_index = partition(data, start, end)
6         # proses partisi sebelah kiri
7         quicksort(data, start, pivot_index-1)
8         # proses partisi sebelah kanan
9         quicksort(data, pivot_index+1, end)
10

```

Implementasi dari fungsi partition() adalah sebagai berikut:

```

1
2 def partition(data, start, end):
3     # ambil salah satu sebagai pivot - (elemen terakhir)
4     pivot = data[end]
5     left = start
6     right = end - 1
7     while left <= right:
8         while data[left] < pivot:
9             left = left + 1
10        while data[right] > pivot:
11            right = right - 1
12        if left <= right:
13            data[left], data[right] = data[right], data[left]
14            left = left + 1
15            right = right - 1
16        # tukar posisi data pada index left dan end
17        data[left], data[end] = data[end], data[left]
18        return left
19

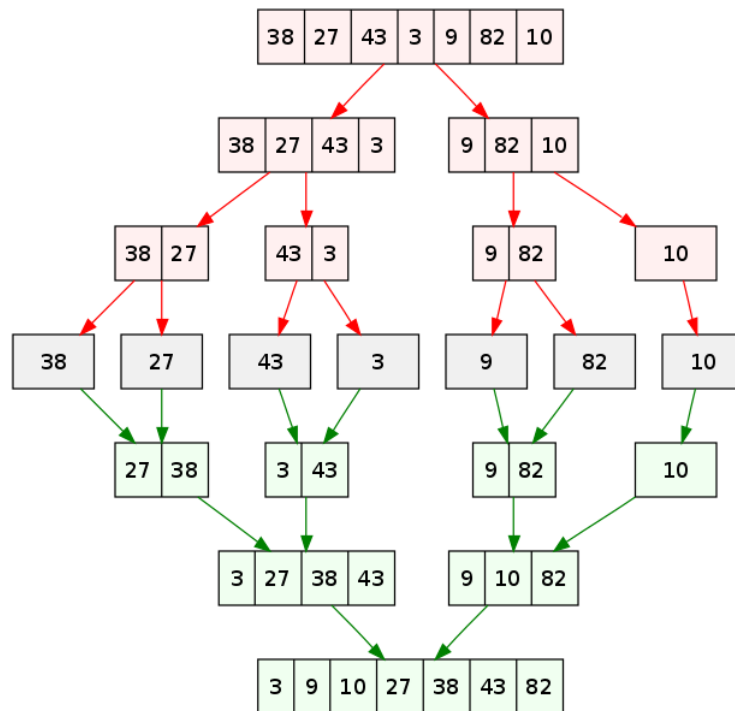
```

10.3.3 Merge Sort

Algoritma Merge Sort merupakan salah satu implementasi dari teknik DaC, yaitu pengurutan data dengan membagi menjadi dua bagian terus-menerus (divide) sampai mendapatkan bagian terkecil yang dapat diurutkan. Kemudian seluruh bagian terkecil tersebut digabungkan secara bertingkat (merge) sehingga seluruh data sudah dalam posisi terurut. Algoritma Merge Sort merupakan algoritma pengurutan yang **stabil**, sehingga jika ada beberapa elemen dengan nilai yang sama, maka posisinya tidak akan berubah setelah diurutkan.

Kinerja dari algoritma Merge Sort ditentukan oleh proses merge, yaitu penggabungan dua bagian data menjadi satu bagian besar yang terurut. Ilustrasi dari algoritma Merge Sort dapat dilihat pada Gambar 10.3.

Algoritma Merge Sort memiliki kompleksitas $O(n \log n)$ di mana tahapan divide adalah $\log n$ (karena data dibagi menjadi dua bagian terus-menerus), kemudian proses merge merupakan proses dengan kompleksitas linier. Kinerja dari algoritma Merge Sort selalu $O(n \log n)$ baik dalam kondisi best case, worst case maupun average case. Hal ini disebabkan oleh proses merge yang selalu sama (linier) dan tidak dipengaruhi oleh nilai elemen atau posisi elemen di dalam data.



Gambar 10.3: Algoritma Merge Sort terdiri dari dua tahapan besar: divide (membagi menjadi dua bagian) dan conquer (merge)

Implementasi algoritma Merge Sort umumnya terbagi menjadi dua langkah utama, yaitu pembagian data menjadi dua bagian dan proses penggabungan kembali setiap bagian yang menghasilkan data dalam posisi terurut. Kode program dari fungsi mergesort() adalah sebagai berikut:

```

1
2 def mergesort(data): # data = list
3     if len(data) > 1:
4         # cari titik tengahnya
5         mid = len(data) // 2
6         # berdasarkan mid, bagi data menjadi dua bagian
7         # pada python bisa menggunakan slicing
8         left = data[:mid]
9         right = data[mid:]
10
11        # bagi lagi sampai bagian terkecil secara rekursif
12        mergesort(left)
13        mergesort(right)
14
15        # merge kiri dan kanan sehingga hasilnya terurut
16        # i => untuk index potongan kiri, j => untuk index potongan kanan
17        i = 0
18        j = 0
19        # k = untuk index list hasil penggabungan kiri dan kanan
20        k = 0

```

```
21
22     while i < len(left) and j < len(right):
23         if left[i] <= right[j]:
24             data[k] = left[i]
25             i += 1
26         else:
27             data[k] = right[j]
28             j += 1
29         k += 1
30
31     # Jika salah satu sudah habis, sisanya tinggal dimasukkan ke list hasil
32     while i < len(left):
33         data[k] = left[i]
34         i += 1
35         k += 1
36
37     while j < len(right):
38         data[k]=right[j]
39         j += 1
40         k += 1
41
```

10.4 Unguided

Bagian unguided akan diberikan secara terpisah di Lab oleh asisten.

dicari langsung berada pada angka hash lokasi penyimpanannya. Namun sering sekali ditemukan hash table yang record-recordnya mempunyai angka hash yang sama (collision). Untuk mengatasi hal ini, maka perlu dilakukan penanganan collision (collision resolution policy) untuk menentukan lokasi record dalam tabel. Umumnya kebijakan resolusi bentrok adalah dengan mencari lokasi tabel yang masih kosong pada lokasi setelah lokasi yang berbentrok

Python menyediakan Dictionary yang menyediakan metode built in untuk penyimpanan data dengan menggunakan hashing. Namun kali ini, kita akan mencoba mengimplementasi hashing secara manual. Adapun method-method yang akan diimplementasikan adalah sebagai berikut:

1. getHash : merupakan implementasi hash function
2. add: menambahkan key pada hash table
3. delete : menghapus key dari hash table
4. get : mendapatkan value berdasarkan key pada hash table
5. print : digunakan untuk menampilkan semua data

Berikut ini adalah contoh kode implementasi hash table pada aplikasi sederhana untuk menyimpan nomor telepon:

```

1 class BukuTelepon:
2     def __init__(self):
3         self.size = 6
4         self.map = [None] * self.size
5
6     def _getHash(self, key):
7         hash = 0
8         for char in str(key):
9             hash += ord(char) # mendapatkan nilai ASCII
10        return hash % self.size
11
12        # menambahkan key pada hash table
13    def add(self, key, value):
14        # periksa apakah key_hash sudah terpakai
15        key_hash = self._getHash(key)
16        # buat pasangan key value
17        key_value = [key, value]
18
19        if self.map[key_hash] is None:
20            self.map[key_hash] = list([key_value])
21            return True
22        else:
23            print("Key Hash ", key_hash, " sudah terisi")
24            return False
25
26    def getPhoneNumber(self, key):
27        key_hash = self._getHash(key)
28        if self.map[key_hash] is not None:
29            for pair in self.map[key_hash]:
30                if pair[0] == key:
31                    return pair[1]
32            print("Key ",key, " tidak ditemukan")
33            return "None"

```



```

34
35     #Menghapus
36     def delete(self, key):
37         key_hash = self._getHash(key)
38         # pastikan key_hash tidak kosong
39         if self.map[key_hash] is None:
40             return False
41         for i in range(0, len(self.map[key_hash])):
42             if self.map[key_hash][i][0] == key:
43                 self.map[key_hash] = None
44                 return True
45         print("Key ",key, " tidak ditemukan")
46         return False
47
48     def print(self):
49         print('---Buku Telepon---')
50         for item in self.map:
51             if item is not None:
52                 print(str(item))

```

Cara penggunaan:

```

1  h = BukuTelepon()
2  h.add('Dendy', '567-8888')
3  h.add('Yuan', '293-6753')
4  h.add('Anton', '333-8233')
5  h.add('Adi', '293-8625')
6  h.add('Dida', '293-8625')
7  h.print()
8  h.delete('Dida')
9  h.print()
10 print('Anton: ' + h.getPhoneNumber('Anton'))

```

Apabila program diatas dieksekusi, maka diketahui bahwa record dengan key Anton tidak ditemukan. Padahal record dengan key Anton sudah diinputkan pada hash table. Bila dianalisa lebih lanjut, gagalnya penambahan data dengan key Anton dikarenakan karena terjadi collision.

11.3.2 Penanganan collision pada hash table

Untuk mengatasi collision pada hash table, kita harus mengimplementasikan mekanisme penanganan collision. Metode penanganan collision yang kita dapat gunakan adalah sebagai berikut :

1. Linear Probing
2. Quadratic probing
3. Double Hashing
4. Separate chaining

Pada contoh berikut akan dilakukan implementasi linear probing, untuk mencegah terjadinya collision saat menginput data pada hash table. Adapun rumus untuk melakukan linear probing adalah sebagai berikut :

$$Slot = (h(key) + i) \% M \quad (11.1)$$

Keterangan : M = jumlah slot hash table, i = Langkah dari probe, $h(\text{key})$ = nilai hash awal (11.2)

Perhatikan implementasi linear probing pada kode berikut, dalam mengatasi terjadinya collision
Cara penggunaan:

```

1 class BukuTelepon:
2     def __init__(self):
3         self.size = 6
4         self.map = [None] * self.size
5
6     def _getHash(self, key):
7         hash = 0
8         for char in str(key):
9             hash += ord(char) # mendapatkan nilai ASCII
10        return hash % self.size
11
12    def _probing(self, key):
13        for index in range(self.size):
14            # probeHash = (self._getHash(key)+index) % self.size
15            probeHash = self._linearProbing(key, index)
16            # valid bila index adalah None atau ber-flag deleted
17            if (self.map[probeHash] is None) or (self.map[probeHash] == 'deleted'):
18                return probeHash
19
20        return None
21
22    # melakukan linear probing
23    def _linearProbing(self, key, index):
24        return (self._getHash(key)+index) % self.size
25
26    # menambahkan key pada hash table
27    def add(self, key, value):
28        # periksa apakah key_hash sudah terpakai
29        key_hash = self._getHash(key)
30        # buat pasangan key value
31        key_value = [key, value]
32
33        if self.map[key_hash] is None:
34            self.map[key_hash] = list([key_value])
35            return True
36        else:
37            key_hash = self._probing(key)
38            if key_hash is None:
39                print("Buku telephone sudah penuh")
40                return False
41
42        self.map[key_hash] = list([key_value])
43        return False

```

```

44
45     def getPhoneNumber(self, key):
46         key_hash = self._getHash(key)
47         if (self.map[key_hash] is not None) and (self.map[key_hash] != 'deleted'):
48             for index in range(self.size):
49                 #mencari dengan melakukan probing
50                 key_hash = self._linearProbing(key, index)
51                 # periksa apakah key adalah data yg akan dihapus
52                 if (self.map[key_hash][0][0] == key):
53                     return self.map[key_hash][0][1]
54
55         print("Key ", key, " tidak ditemukan")
56         return "None"
57
58     def delete(self, key):
59         key_hash = self._getHash(key)
60         if self.map[key_hash] is None:
61             return False
62         for index in range(self.size):
63             #menghapus dengan melakukan linear probing
64             key_hash = self._linearProbing(key, index)
65             # periksa apakah key adalah data yg akan dihapus
66             if (self.map[key_hash][0][0] == key):
67                 print("deleting ", key)
68                 self.map[key_hash] = "deleted"
69                 return True
70
71         print("Key ", key, " tidak ditemukan")
72         return False
73
74     def print(self):
75         print('---Buku Telephone---')
76         for item in self.map:
77             if item is not None:
78                 print(str(item))

```

11.3.3 Latihan Unguided

Bagian unguided akan diberikan secara terpisah di Lab oleh asisten.

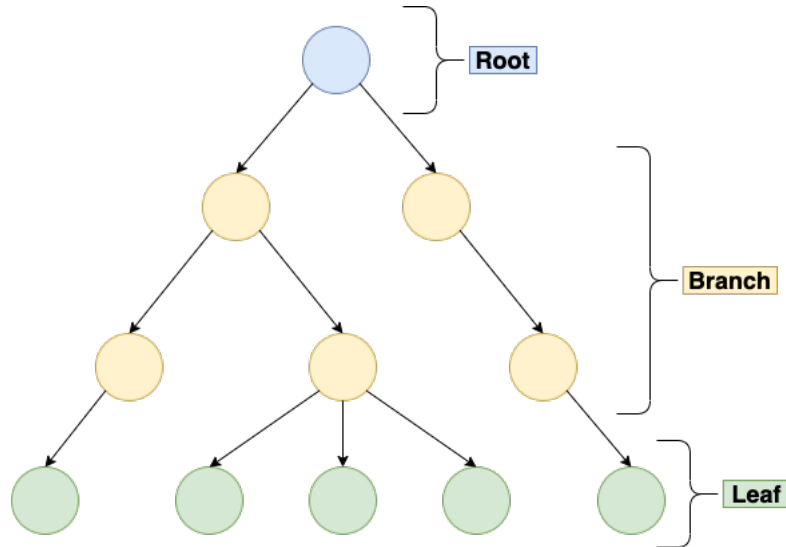


Part 04: Non Linear Data Structures

12	Tree	119
12.1	Tujuan Praktikum	
12.2	Alat dan Bahan	
12.3	Materi	
12.4	Guided	
12.5	Unguided	
13	Binary Tree	125
13.1	Tujuan Praktikum	
13.2	Alat dan Bahan	
13.3	Materi	
13.4	Guided	
13.5	Unguided	
14	Graph	133
14.1	Tujuan Praktikum	
14.2	Alat dan Bahan	
14.3	Graph	
14.4	Implementasi Graph	
14.5	Guided	
14.6	Unguided	
15	Revision History	139
15.1	Daftar Revisi Modul Praktikum	
	Bibliografi	141

3. Leaf/External: Node yang tidak memiliki child

Tree hanya dapat menjelajah satu arah dari parent menuju child, tidak berlaku sebaliknya. Sebuah node hanya dapat memiliki satu parent, tetapi sebuah node dapat memiliki banyak child. Node-node yang memiliki parent yang sama disebut sebagai sibling.



Gambar 12.1: Bentuk Struktur Data Tree

Implementasi Tree diwujudkan dalam bentuk class bernama Node. Sebuah tree terdiri dari banyak object Node yang saling terhubung melalui relasi parent dan child. Berikut adalah template dasar dari class Node:

```

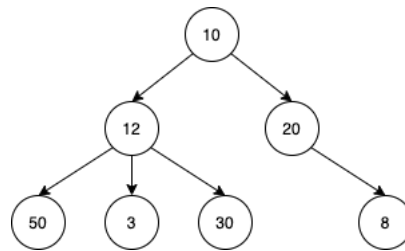
1 class Node:
2     #mendapatkan isi elemen
3     def operator(self):
4
5     #mendapatkan node parent
6     def parent(self):
7
8     #mendapatkan node children
9     def children(self):
10
11    #mengecek apakah node merupakan root
12    def isRoot(self):
13
14    #mengecek apakah node merupakan external/leaf
15    def isExternal(self):
  
```

12.4 Guided

Implementasi Dasar Tree

Buatlah tree dengan struktur sebagai berikut:

Pertama-tama, implementasikan terlebih dahulu template tree seperti pada penjelasan sebelumnya.



Gambar 12.2: Latihan Soal Tree

```

1 class Node:
2     def __init__(self, data, parent):
3         self._data = data
4         self._children = []
5         self._parent = parent
6
7         #menambahkan child
8     def addChild(self, data):
9         self._children.append(data)
10
11        #mendapatkan isi elemen
12    def operator(self):
13        return self._data
14
15        #mendapatkan node parent
16    def parent(self):
17        return self._parent
18
19        #mendapatkan node children
20    def children(self):
21        return self._children
22
23        #mengecek apakah node merupakan root
24    def isRoot(self):
25        return self._parent is None
26
27        #mengecek apakah node merupakan external/leaf
28    def isExternal(self):
29        return len(self._children) == 0
  
```

Pada implementasi di atas ditambahkan sebuah constructor yang menerima parameter berupa 'data' untuk menyimpan nilai pada node, serta 'parent' untuk menyimpan informasi parent dari object node yang bersangkutan. Untuk menguji class di atas, dapat dilakukan dengan membuat masing-masing node-nya terlebih dahulu.

```

1 root = Node(10, None)
2 a = Node(12, root)
3 b = Node(20, root)
  
```

```

4 c = Node(50, a)
5 d = Node(3, a)
6 e = Node(30, a)
7 f = Node(8, b)

```

Kemudian, tambahkan relasi child ke masing-masing parent yang bersangkutan:

```

1 a.addChild(c)
2 a.addChild(d)
3 a.addChild(e)
4 b.addChild(f)
5 root.addChild(a)
6 root.addChild(b)

```

Apabila kita ingin mencetak data yang disimpan pada tree, maka dapat dilakukan dengan memanggil function operator():

```

1 #cetak data pada node d
2 print(d.operator())
3
4 #cetak data dari parent node e
5 print(e.parent().operator())
6
7 #cetak data children dari node a
8 for i in a.children():
9     print(i.operator(), end = ' ')

```

Mengetahui Kedalaman Node (Depth) pada Tree

Setiap node pada tree memiliki tingkat kedalaman tertentu. Untuk mengetahui tingkat kedalaman sebuah node, dapat dilakukan dengan menggunakan fungsi rekursi. Setiap iterasi akan melakukan pengecekan terhadap parent dari node tersebut, selama parent bukan merupakan root maka perulangan akan terus dilakukan.

```

1 def depth(node):
2     if node.isRoot():
3         return 0;
4     else:
5         return 1+depth(node.parent())

```

Coba lakukan pengujian perhitungan depth terhadap beberapa node.

```

1 print(depth(root))
2 print(depth(a))
3 print(depth(e))

```

Mengetahui Tinggi Node (Height) pada Tree

Height adalah tinggi dari sebuah node pada tree. Height dapat dihitung dengan cara mengukur tingkat kedalaman paling dalam dari seluruh keturunan yang dimiliki oleh node tersebut, sehingga dapat dikatakan bahwa nilai height merupakan kebalikan dari depth. Dengan kata lain, node root seharusnya memiliki nilai height paling tinggi dibandingkan dengan node yang lain. Berikut adalah implementasi dari function height:

```

1 def height(node):
2     if node.isExternal():
3         return 0
4     else:
5         h = 0
6         for i in node.children():
7             h = max(h, height(i))
8         return 1+h

```

Coba lakukan pengujian perhitungan height terhadap beberapa node.

```

1 print(height(root))
2 print(height(a))
3 print(height(e))

```

Algoritma Traversal pada Tree

Secara umum, penjelajahan (Traversal) pada tree memiliki dua macam bentuk, yaitu pre-order dan post-order. Pre-order melakukan penjelajahan dimulai dari root dan berakhir pada node leaf yang terletak di posisi paling dalam dan paling kanan dari struktur sebuah tree. Sedangkan penjelajahan post-order dimulai dari node paling dalam dan paling kiri dari sebuah tree, dan akan berakhir pada root dari tree tersebut. Berikut adalah implementasi dari algoritma pre-order dan post-order:

```

1 def preorder(node):
2     print(node.operator(), end = ' ')
3     for i in node.children():
4         preorder(i)
5
6 def postorder(node):
7     for i in node.children():
8         postorder(i)
9     print(node.operator(), end = ' ')

```

Bandingkan hasil dari kedua algoritma tersebut dengan menggunakan node root:

```

1 preorder(root)
2 postorder(root)

```

12.5 Unguided

Bagian unguided akan diberikan secara terpisah di Lab oleh asisten.

[illegible]

13.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat memahami dan mengenal konsep Binary Tree
2. Dapat mengimplementasikan Binary Tree pada Python

13.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.8, 3.9, atau 3.10 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).
6. Moodle UKDW

13.3 Materi

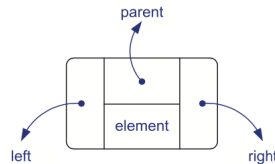
13.3.1 Binary Tree

Binary tree merupakan salah satu turunan dari tree, di mana terdapat beberapa aturan tambahan yaitu:

1. Hanya dapat memiliki maksimal 2 child.

2. Biasanya diimplementasikan sebagai **ordered tree**, yaitu tree di mana node-node di dalamnya disimpan berdasarkan urutan/susunan tertentu.
3. Binary tree dikatakan sempurna (proper) jika seluruh node di dalamnya memiliki 0 atau 2 child. Jika terdapat node yang memiliki hanya 1 child saja maka dapat dikatakan binary tree tersebut tidak sempurna (improper).

Berikut adalah gambar struktur dan template source code class Node dari binary tree:



Gambar 13.1: Struktur Node pada Binary Tree

```

1 class Node:
2     #mendapatkan isi elemen
3     def operator(self):
4
5     #mendapatkan child sebelah kiri
6     def left(self):
7
8     #mendapatkan child sebelah kanan
9     def right(self):
10
11    #mendapatkan node parent
12    def parent(self):
13
14    #mengecek apakah node merupakan root
15    def isRoot(self):
16
17    #mengecek apakah node merupakan external/leaf
18    def isExternal(self):

```

Untuk memudahkan pengelolaan binary tree, maka terdapat satu class bernama BinaryTree untuk memanajemen node=node di dalamnya. Berikut adalah template dari class BinaryTree:

```

1 class BinaryTree:
2     #mendapatkan jumlah node dari tree
3     def size(self):
4
5     #mengecek apakah tree kosong
6     def empty(self):
7
8     #mendapatkan node root
9     def root(self):
10
11    #mendapatkan seluruh node
12    def nodes(self):

```

13.4 Guided

Implementasi Dasar Binary Tree

Binary tree dapat diimplementasikan menggunakan beberapa aturan, salah satunya adalah ordered tree. Ordered tree merupakan tree di mana data-data disimpan berdasarkan urutan tertentu. Pada implementasi kali ini data akan diurutkan secara ascending, di mana data di sebelah kiri lebih kecil dibandingkan dengan data pada node, sedangkan data di sebelah kanan lebih besar dibandingkan node. Berikut adalah implementasi class Node pada binary tree:

```

1 class Node:
2     #constructor
3     def __init__(self, data, parent):
4         self._data = data
5         self._parent = parent
6         self._left = None
7         self._right = None
8
9     #menambah data
10    def insert(self, data):
11        if data < self.operator():
12            if self.left() is None:
13                self._left = Node(data, self)
14            else:
15                self.left().insert(data)
16        elif data > self.operator():
17            if self.right() is None:
18                self._right = Node(data, self)
19            else:
20                self.right().insert(data)
21        else:
22            return False #jika tidak berhasil menambah data
23            return True #jika berhasil menambah data
24
25    #mendapatkan isi elemen
26    def operator(self):
27        return self._data
28
29    #mendapatkan child sebelah kiri
30    def left(self):
31        return self._left
32
33    #mendapatkan child sebelah kanan
34    def right(self):
35        return self._right
36
37    #mendapatkan node parent
38    def parent(self):
39        return self._parent
40
41    #mengecek apakah node merupakan root

```

```

42     def isRoot(self):
43         return self._parent is None
44
45     #mengecek apakah node merupakan external/leaf
46     def isExternal(self):
47         return self._left is None and self._right is None

```

Class Node di atas akan digunakan oleh class BinaryTree di bawah ini. Implementasi method nodes() menggunakan metode in order traversal, yaitu metode penjelajahan yang dimulai dari node paling kiri dan berakhir di node yang paling kanan, sehingga node root akan diakses di tengah-tengah proses penjelajahan. Dengan demikian, apabila in order traversal diimplementasikan pada ordered tree, maka outputnya adalah data yang urut sesuai aturan pengurutan.

```

1  class BinaryTree:
2      #constructor
3      def __init__(self):
4          self._root = None
5          self._size = 0
6
7      #menambah data
8      def add(self, data):
9          if self._root is None:
10             self._root = Node(data, None)
11             self._size+=1
12          else:
13             if self._root.insert(data):
14                 self._size+=1
15
16      #mendapatkan jumlah node dari tree
17      def size(self):
18          return self._size
19
20      #mengecek apakah tree kosong
21      def empty(self):
22          return self._size == 0
23
24      #mencetak seluruh node
25      def nodes(self):
26          self.inorder(self._root)
27
28      #inorder traversal
29      def inorder(self, node):
30          if node is not None:
31              self.inorder(node.left())
32              print(node.operator(), end = ' ')
33              self.inorder(node.right())

```

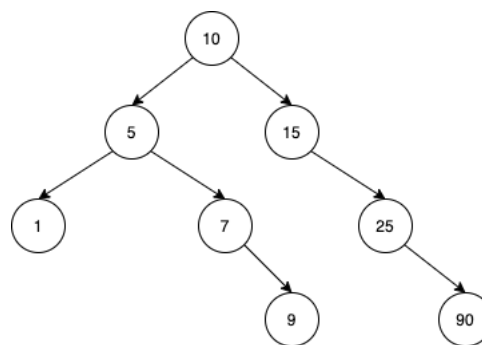
Lakukan pengujian source code di atas dengan data seperti berikut:

```

1 tree = BinaryTree()
2 tree.add(10)
3 tree.add(5)
4 tree.add(7)
5 tree.add(1)
6 tree.add(15)
7 tree.add(9)
8 tree.add(25)
9 tree.add(90)
10 tree.nodes()

```

Struktur dari tree di atas menjadi seperti berikut:



Gambar 13.2: Struktur Binary Tree Hasil Pengujian

Pencarian pada Binary Tree (Binary Search Tree)

Karena binary tree diimplementasikan secara terurut (ordered), maka kita dapat menerapkan beberapa algoritma, salah satunya adalah binary search. Binary search merupakan metode pencarian pada tree yang didasarkan pada perbandingan node di sebelah kiri dan kanan. Tambahkan method `search()` dan `binarySearch()` di dalam class `BinaryTree`:

```

1 #mencari data
2 def search(self, value):
3     return self.binarySearch(self._root, value) is not None
4
5 #algoritma binary search
6 def binarySearch(self, node, value):
7     if node is None or node.operator() == value:
8         return node
9     elif value < node.operator():
10        return self.binarySearch(node.left(), value)
11    else:
12        return self.binarySearch(node.right(), value)

```

Kemudian lakukan pengujian seperti berikut:

```

1 print(tree.search(25))
2 print(tree.search(8))

```

Update Data: Expand dan Remove

Selain operasi penambahan data, terdapat operasi lain untuk memperbarui data yaitu `expandExternal()` dan `removeAboveExternal()`. Operasi `expandExternal()` bertujuan untuk mengubah `external` node menjadi `internal` node dengan menambahkan `child` di sisi kiri dan kanan node tersebut. Sedangkan `removeAboveExternal()` adalah operasi untuk menghapus sebuah node beserta `parent`-nya.

Pada implementasinya, node yang akan di-expand maupun di-remove akan dicari dulu berdasarkan nilai yang disimpan di dalamnya. Khusus untuk method `expandExternal`, kita akan mengimplementasikan dengan menambahkan nilai 1 angka lebih rendah di node sebelah kiri, dan 1 angka lebih tinggi di node sebelah kanan.

Untuk mengakomodasi operasi tersebut, maka perlu ditambahkan beberapa method pada class `Node` karena kita harus mengubah property `left`, `right`, dan `parent` saat operasi dilakukan:

```

1  #mengeset node left
2  def setLeft(self, node):
3      self._left = node
4
5  #mengeset node right
6  def setRight(self, node):
7      self._right = node
8
9  #mengeset node parent
10 def setParent(self, node):
11     self._parent = node

```

Kemudian, implementasikan kedua method `expandExternal()` dan `removeAboveExternal()` pada class `BinaryTree`:

```

1  #mencari data
2  def expandExternal(self, value):
3      node = self.binarySearch(self._root, value)
4      if node is not None and node.isExternal():
5          node.insert(node.operator()-1)
6          node.insert(node.operator()+1)
7          self._size+=2
8
9  #mencari data
10 def removeAboveExternal(self, value):
11     node = self.binarySearch(self._root, value)
12     if node is not None:
13         sibling = node.parent().left() if node == node.parent().right() else node.parent()
14         if node.parent() == self._root:
15             self._root = sibling
16             sibling.setParent(None)
17         else:
18             if node.parent() == node.parent().parent().left():
19                 node.parent().parent().setLeft(sibling)
20             else:

```

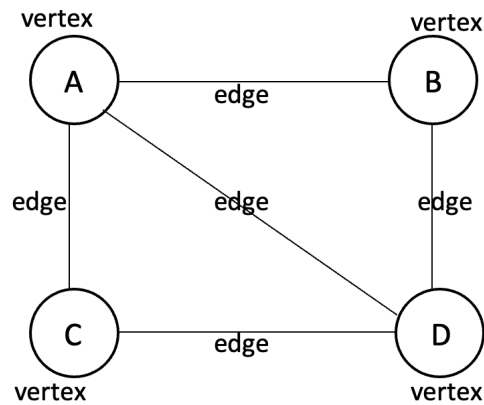
```
21         node.parent().parent().setRight(sibling)
22     self._size-=2
```

Lakukan pengujian pada operasi tersebut:

```
1 tree.expandExternal(9)
2 tree.nodes()
3 tree.removeAboveExternal(7)
4 tree.nodes()
```

13.5 Unguided

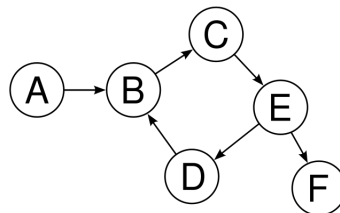
Bagian unguided akan diberikan secara terpisah di Lab oleh asisten.



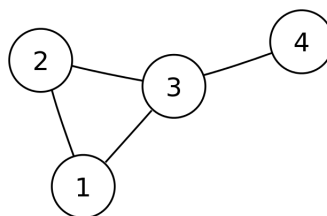
Gambar 14.1: Struktur Data Graph

sehingga dapat dikatakan bahwa sebuah node memiliki kedudukan yang lebih tinggi (parent), lebih rendah (child), atau setara (sibling) dengan node yang lain. Sedangkan pada graph, kedudukan setiap node (vertex) adalah setara, sehingga tidak ada node yang memiliki kedudukan lebih tinggi atau lebih rendah dibanding node yang lain.

Terdapat beberapa jenis graph, di antaranya adalah graph berarah dan graph tidak berarah. Graph berarah merupakan graph yang setiap edge-nya memiliki arah, misalnya jika dari vertex A terdapat edge satu arah untuk menuju vertex B, maka dari vertex B kita tidak bisa pergi menuju vertex A. Namun pada graph tidak berarah, apabila dari vertex A terdapat edge menuju vertex B, maka berlaku sebaliknya dari vertex B dapat pergi menuju vertex A. Dengan kata lain, graph tidak berarah adalah graph yang seluruh edge-nya memiliki dua arah.

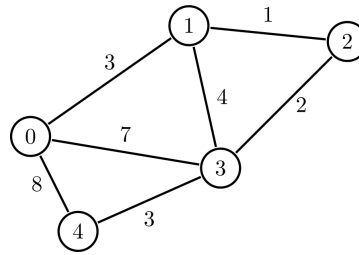


Gambar 14.2: Graph Berarah



Gambar 14.3: Graph Tidak Berarah

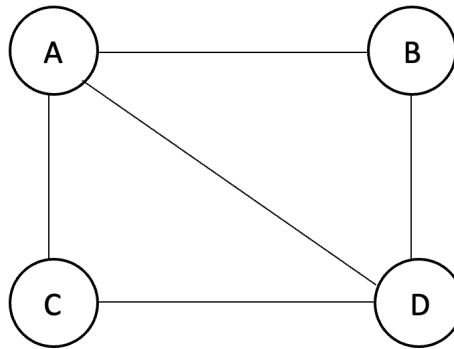
Selain ditinjau dari arahnya, terdapat graph jenis lain yang ditinjau dari bobot-nya. Artinya, setiap edge pada graph memiliki nilai bobot masing-masing untuk menandakan jarak atau beban yang harus ditempuh saat melakukan perjalanan dari vertex A menuju vertex B.



Gambar 14.4: Graph Berbobot

14.4 Implementasi Graph

Graph dapat diimplementasikan menggunakan kombinasi antara list dan dictionary, dengan asumsi setiap vertex memiliki nilai yang unik. Misalnya pada contoh graph tidak berarah di bawah ini:



Gambar 14.5: Contoh Graph

Terdapat 4 buah vertex yaitu: a, b, c, d dan terdapat 5 buah edge yaitu: ab, ac, ad, bd, cd. Maka kita dapat membuat struktur dictionary-nya menjadi seperti berikut ini:

```

1 graph = {
2     "a": ["b", "c", "d"],
3     "b": ["a", "d"],
4     "c": ["a", "d"],
5     "d": ["a", "b"]
6 }
```

14.5 Guided

Pada latihan ini kita akan membuat ADT dari graph beserta beberapa fungsi di dalamnya, yaitu fungsi untuk menambah vertex, menambah edge, serta find path seperti berikut ini.

```

1 class Graph:
2     def __init__(self):
3         #constructor
4
5     def addVertex(self, key):
```

```

6         #menambah vertex
7
8     def vertex(self):
9         #mencetak seluruh vertex
10
11     def addEdge(self, x, y):
12         #menambah edge antara vertex x dan y
13
14     def edge(self):
15         #mencetak seluruh edge yang ada
16
17     def findPath(self, x, y):
18         #mencari jalur dari vertex x menuju y

```

Pertama-tama implementasikan constructor dengan membuat dictionary kosong untuk menampung seluruh informasi graph.

```

1 def __init__(self):
2     self._data = {}

```

Kemudian, saat fungsi addVertex() dipanggil, tambahkan key baru pada dictionary dengan nilai di dalamnya berupa set kosong. Sedangkan fungsi vertex() bertujuan untuk mencetak seluruh nilai vertex yang ada pada Graph.

```

1 def addVertex(self, key):
2     if key not in self._data:
3         self._data[key] = set()
4
5 def vertex(self):
6     for key, value in self._data.items():
7         print(key, end = ' ')
8     print()

```

Pada fungsi addEdge, tuliskan baris program seperti di bawah ini. Fungsi tersebut berjalan dengan asumsi Graph bertipe tidak berarah, artinya jika kita dapat pergi dari vertex x menuju vertex y, maka berlaku sebaliknya.

```

1 def addEdge(self, x, y):
2     if x in self._data and y in self._data:
3         self._data[x].add(y)
4         self._data[y].add(x) #hanya digunakan jika graph tidak berarah
5
6 def edge(self):
7     listEdge = set()
8     for key, value in self._data.items():
9         for item in self._data[key]:
10            if key+item not in listEdge and item+key not in listEdge:

```



```

11         listEdge.add(key+item)
12     for item in listEdge:
13         print(item, end = ' ')
14     print()

```

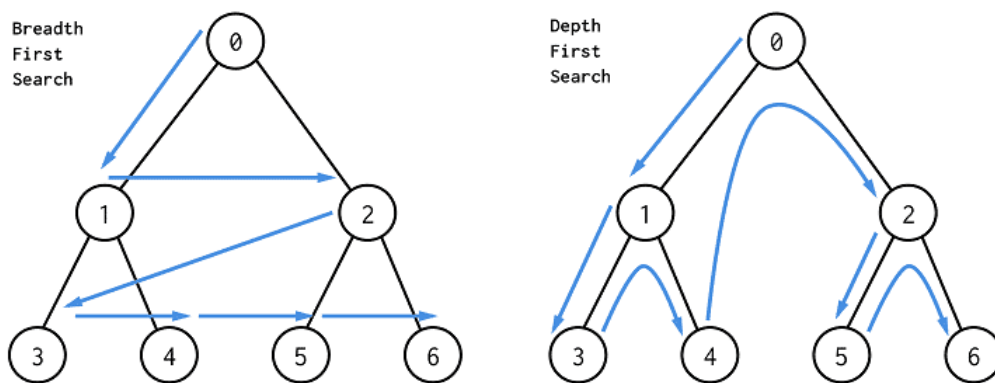
Untuk mencoba graph di atas, gunakan kode berikut ini:

```

1 graph = Graph()
2 graph.addVertex('a')
3 graph.addVertex('b')
4 graph.addVertex('c')
5 graph.addVertex('d')
6 graph.addEdge('a', 'b')
7 graph.addEdge('a', 'c')
8 graph.addEdge('a', 'd')
9 graph.addEdge('b', 'd')
10 graph.addEdge('c', 'd')
11 graph.vertex()
12 graph.edge()

```

Pada tahap berikutnya kita akan mencoba mencari jalur (path) yang diperlukan untuk pergi dari vertex x menuju vertex y. Vertex x dan vertex y belum tentu dihubungkan langsung oleh sebuah edge, sehingga perlu diperlukan kombinasi dari banyak edge agar vertex x dapat terhubung dengan vertex y melalui vertex-vertex lain.



Gambar 14.6: BFS dan DFS

Terdapat beberapa algoritma untuk pencarian jalur, dua algoritma yang paling sederhana adalah Breadth First Search (BFS) dan Depth First Search (DFS). Kedua algoritma tersebut memiliki konsep seperti Tree, di mana vertex asal akan dijadikan sebagai root, sedangkan vertex yang terhubung dengan root tersebut akan dijadikan sebagai child. Pada BFS, pencarian dilakukan secara horizontal pada setiap level kedalaman, sehingga pencarian di level yang lebih dalam hanya akan dilakukan jika pencarian pada level sebelumnya telah selesai dijelajahi. Sedangkan pada DFS, pencarian dilakukan secara vertikal hingga ke titik paling dalam. Kedua algoritma tersebut termasuk jenis pencarian buta, sehingga jalur yang akan didapatkan belum tentu jalur yang paling pendek atau efisien. Agar dapat menemukan jalur yang paling efisien, terdapat algoritma lain yang lebih baik seperti Dijkstra atau A-star, namun dengan tingkat kompleksitas yang lebih rumit.

Pada fungsi `findpath`, kita akan mengimplementasikannya menggunakan algoritma DFS seperti contoh berikut ini:

```
1 def findPath(self, x, y):
2     visited = []
3     self.dfs(x, y, visited)
4
5 def dfs(self, node, y, visited):
6     visited.append(node)
7     if node == y:
8         print(visited)
9     else:
10        for item in self._data[node]:
11            if item not in visited:
12                self.dfs(item, y, visited)
```

Dengan data yang sama seperti sebelumnya, coba lakukan pencarian jalur dari vertex b menuju c.

14.6 Unguided

Bagian unguided akan diberikan secara terpisah di Lab oleh asisten.

15. Revision History

15.1 Daftar Revisi Modul Praktikum

15.1.1 Semester Gasal 2022/2023

Pada semester ini dilakukan perbaikan penulisan pada beberapa bagian bab oleh Antonius Rachmat. Editorial juga disesuaikan untuk semester dan pengajar.

Penulis: Yuan Lukito (Penulis pertama), Kristian Adi Nugraha (Penulis kedua), Antonius Rachmat (Penulis ketiga).

15.1.2 Semester Gasal 2021/2022

Pada semester ini ditambahkan dua bab baru, yaitu materi Sorting (Quick sort dan Merge sort) yang ditulis oleh Yuan Lukito dan materi Graph yang ditulis oleh Kristian Adi Nugraha.

Penulis: Yuan Lukito (Penulis pertama) dan Kristian Adi Nugraha (Penulis kedua).

15.1.3 Semester Gasal 2020/2021

Merupakan semester pertama modul praktikum ini digunakan. Berisi 12 materi yang disusun oleh Yuan Lukito, Antonius Rachmat, Kristian Adi Nugraha dan I Kadek Dendy Prtha.

Penulis: Yuan Lukito (Penulis pertama), Antonius Rachmat C, Kristian Adi Nugraha dan I Kadek Dendy Prtha (Penulis kedua, ketiga dan ke-empat).

- [1] Goodrich, M. T, Tamassia, R., Goldwasser, M. H. (2013) *Data Structures and Algorithms in Python*. Wiley
- [2] Python Software Foundation (2021) *Python Documentation* . Python Software Foundation