

Pertemuan 1

Konsep Struktur Data & Array

STRUKTUR DATA

Struktur Data adalah : suatu koleksi atau kelompok data yang dapat dikarakteristikan oleh organisasi serta operasi yang didefinisikan terhadapnya.

Pemakaian Struktur Data yang tepat didalam proses pemrograman, akan menghasilkan Algoritma yang lebih jelas dan tepat sehingga menjadikan program secara keseluruhan lebih sederhana.

Pada garis besarnya, Data dapat dikategorikan menjadi :

A. Type Data

Type Data	Contoh	Penjelasan
Boolean	<code>True</code> atau <code>False</code>	Menyatakan benar <code>True</code> yang bernilai <code>1</code> , atau salah <code>False</code> yang bernilai <code>0</code>
String	<code>"Ayo belajar Python"</code>	Menyatakan karakter/kalimat bisa berupa huruf angka, dll (diapit tanda <code>"</code> atau <code>'</code>)
Integer	<code>25</code> atau <code>1209</code>	Menyatakan bilangan bulat
Float	<code>3.14</code> atau <code>0.99</code>	Menyatakan bilangan yang mempunyai koma
Hexadecimal	<code>9a</code> atau <code>1d3</code>	Menyatakan bilangan dalam format heksa (bilangan berbasis 16)
Complex	<code>1 + 5j</code>	Menyatakan pasangan angka real dan imajiner
List	<code>['xyz', 786, 2.23]</code>	Data untaian yang menyimpan berbagai tipe data dan isinya bisa diubah-ubah
Tuple	<code>('xyz', 768, 2.23)</code>	Data untaian yang menyimpan berbagai tipe data tapi isinya tidak bisa diubah
Dictionary	<code>{'nama': 'adi', 'id': 2}</code>	Data untaian yang menyimpan berbagai tipe data berupa pasangan penunjuk dan nilai

B. Struktur Data

Terdiri dari :

1. Struktur Data Sederhana

Misalnya Array dan Record

2. Struktur Data Majemuk

Terdiri dari :

a. Linier

Misalnya : Stack, Queue dan Linear Linked List.

b. Non Linier

Misalnya : Pohon (Tree), Pohon Biner (Binary Tree), Pohon Cari Biner (Binary Search Tree), General Tree serta Graph.

Array Dimensi 1 & 2

- ❖ Array / Larik : Struktur Data Sederhana yang dapat didefinisikan sebagai pemesanan alokasi memory sementara pada komputer.
- ❖ Array dapat didefinisikan sebagai suatu himpunan hingga elemen yang terurut dan homogen.
- ❖ Terurut : Dapat diartikan bahwa elemen tersebut dapat diidentifikasi sebagai elemen pertama, elemen kedua dan seterusnya sampai elemen ke-n.
- ❖ Homogen : Adalah bahwa setiap elemen dari sebuah Array tertentu haruslah mempunyai type data yang sama.

Definisi Array

- ❖ Sebuah Array dapat mempunyai elemen yang seluruhnya berupa integer atau character atau String bahkan dapat pula terjadi suatu Array mempunyai elemen berupa Array.

- ❖ **Karakteristik Array :**
 1. Mempunyai batasan dari pemesanan alokasi memory (Bersifat Statis)
 2. Mempunyai Type Data Sama (Bersifat Homogen)
 3. Dapat Diakses Secara Acak

Definisi Array (Lanjutan)

- ❖ **3 Hal** yang harus diketahui dalam mendeklarasikan array :
 - a. Type data array
 - b. Nama variabel array
 - c. Subskrip / index array

- ❖ **Jenis Array** (yang akan dipelajari) adalah :
 - a. Array Dimensi Satu (One Dimensional Array)
 - b. Array Dimensi Dua (Two Dimensional Array)
 - c. Array Dimensi Tiga (Thee Dimensional Array)

Apa itu Array dan mengapa harus menggunakannya?

Array pada dasarnya adalah struktur data (Data Structures) yang dapat menampung lebih dari satu nilai pada satu waktu. Array adalah kumpulan atau rangkaian elemen yang berurutan dan memiliki jenis yang sama. Array dapat melakukan loop dengan mudah dan mengambil nilai yang diperlukan hanya dengan menentukan nomor indeks. Array juga bisa berubah (berubah), oleh karena itu, kita dapat melakukan berbagai manipulasi sesuai kebutuhan.

Dengan menggunakan array pada Python maka kita dapat menghemat banyak waktu. Seperti yang disebutkan sebelumnya, array membantu kita mengurangi ukuran keseluruhan kode yang kita buat, sementara Python membantu kita menghilangkan sintaks yang bermasalah, tidak seperti bahasa lain. Misalnya: Jika kita ingin menyimpan bilangan bulat dari 1100, kita tidak akan dapat mengingat 100 nama variabel secara eksplisit, oleh karena itu, kita dapat menyimpannya dengan mudah menggunakan Array.

Apa Perbedaan Array Dengan List?

- Apabila Array diartikan sebagai struktur data yang dapat menampung lebih dari satu nilai dalam satu waktu, lalu apa perbedaannya dengan list? Dan bagaimana hubungannya dengan data structures lain seperti series dan data frame?
- Pada dasarnya, array mirip seperti list namun tidak sepenuhnya sama. Ada beberapa kesamaan seperti sama-sama menampung lebih dari satu nilai dalam satu waktu, sama-sama dapat diiterasi dan diindeks. Perbedaan antara List dan array adalah fungsi yang dapat dilakukan padanya seperti misalnya ketika kita ingin membagi array dengan suatu nilai (misal, 2), hasilnya akan dicetak sesuai permintaan tetapi dalam kasus list, python akan menampilkan pesan error.

Metode Array yang terdapat pada Python

Method	Deskripsi
<code>append()</code>	Menambahkan elemen di akhir list
<code>clear()</code>	Menghapus semua elemen dari list
<code>copy()</code>	Mengembalikan salinan list
<code>count()</code>	Mengembalikan jumlah elemen dengan nilai yang ditentukan
<code>extend()</code>	Tambahkan elemen list (atau iterable), ke akhir list saat ini
<code>index()</code>	Mengembalikan indeks elemen pertama dengan nilai yang ditentukan
<code>insert()</code>	Menambahkan elemen pada posisi yang ditentukan
<code>pop()</code>	Menghapus elemen pada posisi yang ditentukan
<code>remove()</code>	Menghapus item dengan nilai yang ditentukan
<code>reverse()</code>	Membalik urutan list
<code>sort()</code>	Mengurutkan list

Array Dimensi Satu

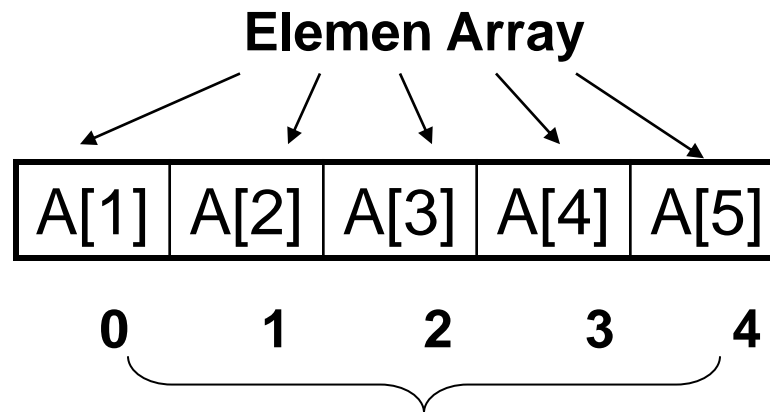
1. ARRAY DIMENSI SATU (One Dimensional Array)

Dapat disebut juga dengan istilah vektor yang menggambarkan data dalam suatu urutan

Deklarasi : Nama_Array [jumlah_elemen];

Misalnya : A[5];

Penggambaran secara Logika :



Subscript / Index

Nomor indeks array di mulai dari 0

Array Dimensi Satu (Lanjutan)

```
1
2 # Online Python - IDE, Editor, Compiler, Interpreter
3
4 import numpy as np
5
6 arr = np.array([4, 3, 2, 1])
7
8 print(arr[0])
9
```

Ln: 8, Col: 14



Run



Share

Command Line Arguments



4

```
1 mobil = ["Ford", "Volvo", "BMW"]
2
3 x = mobil[0]
4
5 print(x)
6
```

Ln: 5, Col: 9



Run



Share

Command Line Arguments



Ford

Array Dimensi Satu (Lanjutan)

Rumus untuk menentukan jumlah elemen dalam Array :

$$\sum_{i=1}^n \pi \text{ (Elemen Array)}$$

π = Perkalian dari elemen sebelumnya
(untuk array dimensi dua & tiga)

Contoh :

Suatu Array A dideklarasikan sbb :

A[10]; maka jumlah elemen Array dimensi satu tersebut
adalah = 10

PEMETAAN (MAPPING) ARRAY DIMENSI SATU KE STORAGE

$$\text{Rumus : } @A[i] = B + (i - 1) * L$$

Dimana : $@A[i]$: Posisi Array yg dicari

B : Posisi awal index di memory komputer

i : Subkrip atau indeks array yg dicari

L : Ukuran / Besar memory suatu type data

Contoh :

Suatu Array A dideklarasikan sebagai berikut :

int A[5]; dengan alamat awal index berada di 0011_(H) dan
ukuran memory type data integer = 2

Tentukan berapa alamat array A[3] ?

PEMETAAN (MAPPING) ARRAY DIMENSI SATU KE STORAGE (Lanjutan)

Rumus : $@A[i] = B + (i - 1) * L$

Diketahui :

$$@A[i] = A[3]$$

$$B = 0011 \text{ (H)}$$

$$i = 3$$

$$L = 2$$

Penyelesaian :

$$A[3] = 0011 \text{ (H)} + (3 - 1) * 2$$

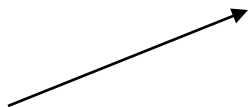
$$= 0011 \text{ (H)} + 4 \text{ (D)}$$

$$= 0011 \text{ (H)} + 4 \text{ (H)}$$

$$= 0015 \text{ (H)}$$

4 Desimal = 4 Hexa

0	1	2	3	4
A[1]	A[2]	A[3]	A[4]	A[5]
0011	0013	0015	0017	0019



KONVERSI BILANGAN

1. Decimal adalah bilangan berbasis sepuluh yang terdiri dari 0, 1, 2, 3, 4, 5, 6, 7, 8, dan 9
2. Hexadecimal adalah bilangan berbasis enam belas yang terdiri dari 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, dan F

Tabel di bawah adalah contoh **konversi bilangan** Decimal, dan Hexadecimal

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Contoh KONVERSI ANTAR BILANGAN

Konversi Bilangan Decimal ke Hexadecimal

Contoh $254_{(10)} = \dots\dots(16)$

Caranya dengan membagi bilangan tersebut dengan enam belas sampai bilangan tersebut tidak bisa lagi dibagi enam belas (kurang dari enam belas) dengan mencatat setiap sisa pembagian.

$254 : 16 = 15$ sisa 14 atau E (lihat tabel di atas)

$15 : 16 =$ sisa 15 atau F (lihat tabel di atas)

Jadi $254_{(10)} = FE_{(16)}$ diurutkan dari sisa pembagian terakhir.

Array Dimensi Dua

2. ARRAY DIMENSI DUA (Two Dimensional Array)

Sering digunakan dalam menterjemahkan matriks pada pemrograman.

Deklarasi : Nama_Array [jumlah elemen baris] [jumlah elemen kolom];

Misalnya : A[3][2];

Penggambaran secara Logika :

	0	1
0		
1		
2		

Array Dimensi Dua (Lanjutan)

Menentukan jumlah elemen dalam Array dimensi dua:

$$\sum_{i=1}^n \pi$$

(Elemen array)

π = Perkalian dari elemen sebelumnya
(untuk array dimensi dua & tiga)

Contoh :

Suatu Array X dideklarasikan sbb :

$X[4][3];$

maka jumlah elemen Array dimensi dua tersebut adalah :

$$(4) * (3) = 12$$

PEMETAAN (MAPPING) ARRAY DIMENSI DUA KE STORAGE

Terbagi Dua cara pandang (representasi) yang berbeda :

1. Secara Kolom Per Kolom (Coloumn Major Order/CMO)

$$@M[i][j] = M[0][0] + \{(j - 1) * K + (i - 1)\} * L$$

2. Secara Baris Per Baris (Row Major Order / RMO)

$$@M[i][j] = M[0][0] + \{(i - 1) * N + (j - 1)\} * L$$

Keterangan :

@M[i][j] = Posisi Array yg dicari, M[0][0] = Posisi alamat awal index array, i = Baris, j = kolom, L = Ukuran memory type data

K = Banyaknya elemen per kolom, N = Banyaknya elemen per baris

Penggambaran secara logika

Misal : $M[3][2]$;
(Array dengan 3 Baris & 2 Kolom)

	0	1
0		
1		
2		

Berdasarkan Cara pandang :

1. Kolom Per Baris (Row Major Order / RMO)

$M[0,0]$	$M[0,1]$	$M[1,0]$	$M[1,1]$	$M[2,0]$	$M[2,1]$
----------	----------	----------	----------	----------	----------

Jumlah elemen per baris = 2

2. Baris Per Kolom (Coloumn Major Order / CMO)

$M[0,0]$	$M[1,0]$	$M[2,0]$	$M[0,1]$	$M[1,1]$	$M[2,1]$
----------	----------	----------	----------	----------	----------

Jumlah elemen per kolom = 3

Contoh Pemetaan :

Suatu Array X dideklarasikan sebagai berikut :
Float X[4][3], dengan alamat index X[0][0] berada
di $0011_{(H)}$ dan ukuran type data float = 4

Tentukan berapa alamat array X[3][2]
berdasarkan cara pandang baris dan kolom ?

	0	1	2	← index
0	0011 _(H)			
1				
2				
3				

↑
index

Contoh Pemetaan (Lanjutan)

Penyelesaian :

Secara Baris Per Baris (Row Major Oder / RMO)

$$@M[i][j] = @M[0][0] + \{(i - 1) * N + (j - 1)\} * L$$

$$X[3][2] = 0011_{(H)} + \{(3 - 1) * 3 + (2 - 1)\} * 4$$

$$= 0011_{(H)} + 28_{(D)} \quad \longrightarrow \quad 1C_{(H)}$$

$$= 0011_{(H)} + 1C_{(H)}$$

$$= 002D_{(H)}$$

Penyelesaian :

Secara Kolom Per Kolom (Coloumn Major Oder / CMO)

$$@M[i][j] = @M[0][0] + \{(j - 1) * K + (i - 1)\} * L$$

$$X[3][2] = 0011(H) + \{(2 - 1) * 4 + (3 - 1)\} * 4$$
$$= 0011(H) + 24 (D) \quad \xrightarrow{\quad} \quad 18 (H)$$

$$= 0011(H) + 18 (H)$$

$$= 0029(H)$$

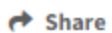
Contoh Program Array Dua Dimensi

```
1 import numpy as np
2
3 arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
4
5 print('3rd element on 2nd dim: ', arr[1, 2])
6
```

Ln: 5, Col: 45



Run



Share

Command Line Arguments



3rd element on 2nd dim: 8

```
1 array_input = [ [10,12,14] , [0,1,2] ]
2 print(array_input[0]) # printing elements of row 0
3 print(array_input[1]) # printing elements of row 1
4
```

Ln: 3, Col: 51



Run



Share

Command Line Arguments



[10, 12, 14]



[0, 1, 2]

<https://www.online-python.com/>

Latihan

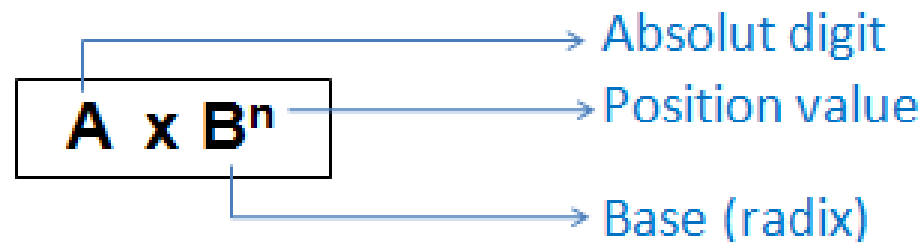
1. Suatu array A dideklarasikan sbb :
A[50] dengan alamat awal berada di 0011(H).
Tentukan berapa alamat array A[20] dan A[40]
dengan tipe data int?
2. Suatu array X dideklarasikan sbb :
X[4][5] dengan alamat awal berada pada
0011(H). Tentukan berapa alamat array X[4][3],
berdasarkan cara pandang baris dan kolom,
dengan tipe data float?

Pertemuan 2

SISTEM BILANGAN

I. Konsep Dasar Sistem Bilangan

- Sistem bilangan adalah suatu cara untuk mewakili besaran dari suatu item fisik.
- Konsep dasar sistem bilangan dikarakteristikkan oleh basis (radix), absolute digit dan posisi (*place*) value, yang dituliskan:



- Basis yang digunakan sistem bilangan tergantung dari jumlah nilai bilangan yang dipergunakan.

Konsep Dasar Sistem Bilangan (Lanjutan)

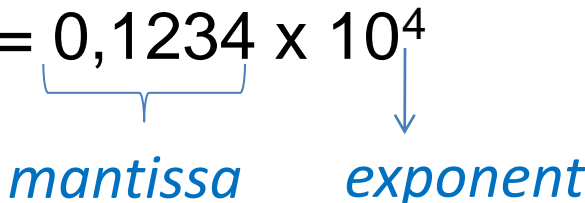
Sistem bilangan yang sering digunakan adalah:

- Sistem bilangan desimal
- Sistem bilangan biner
- Sistem bilangan oktal
- Sistem bilangan hexadesimal

1. Sistem Bilangan Desimal

- Sistem bilangan desimal menggunakan basis 10 (*deca*)
- Menggunakan 10 macam simbol bilangan berbentuk digit angka: 0,1,2,3,4,5,6,7,8,9
- Dasar penulisan:

$$A \times 10^n$$

- Bentuk nilai desimal dapat berupa *integer* (bilangan bulat) dan pecahan
- Dapat ditulis dalam bentuk eksponensial yaitu ditulis dengan *mantissa* dan *exponent*.
- Contoh: $1234 = 0,1234 \times 10^4$


Sistem Bilangan Desimal (Lanjutan)

- Penulisan *base/radix* dituliskan setelah absolut digit, yaitu A_{10} , atau $A(D)$.

Dalam hal ini yang dituliskan adalah A_{10}

- Contoh nilai 4352_{10} dan $762,15_{10}$ dapat diartikan:

$$\begin{array}{rcl} 4 \times 10^3 & = & 4000 \\ 3 \times 10^2 & = & 300 \\ 5 \times 10^1 & = & 50 \\ 2 \times 10^0 & = & \underline{2} + \\ & & 4352 \end{array}$$

$$\begin{array}{rcl} 7 \times 10^2 & = & 700 \\ 6 \times 10^1 & = & 60 \\ 2 \times 10^0 & = & 2 \\ 1 \times 10^{-1} & = & 0,1 \\ 5 \times 10^{-2} & = & \underline{0,05} + \\ & & 762,15 \end{array}$$

2. Sistem Bilangan Biner

- Sistem bilangan biner menggunakan basis 2 (*binary*)
- Menggunakan 2 macam simbol bilangan berbentuk digit angka: 0 dan 1
- Penulisan *base/radix* dituliskan setelah absolut digit, yaitu A_2 atau $A(B)$. Dalam hal ini yang dituliskan adalah A_2
- Dasar penulisan: $A \times 2^n$
- Contoh penulisan: 1001 0011₂

3. Sistem Bilangan Oktal

- Sistem bilangan oktal menggunakan basis 8 (*octal*)
- Menggunakan 8 macam simbol bilangan berbentuk digit angka: 0,1,2,3,4,5,6,7
- Penulisan *base/radix* dituliskan setelah absolut digit, yaitu A_8 atau $A(O)$. Dalam hal ini yang dituliskan adalah A_8
- Dituliskan: $A \times 8^n$
- Contoh penulisan: 347_8

4. Sistem Bilangan Hexadesimal

- Sistem bilangan hexadesimal menggunakan basis 16 (*hexa*)
- Menggunakan 16 macam simbol bilangan berbentuk digit angka: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- Penulisan *base/radix* dituliskan setelah absolut digit, yaitu A_{16} atau A(H). Dalam hal ini yang dituliskan adalah A_{16}
- Dituliskan: $A \times 16^n$
- Contoh penulisan: $A78_{16}$

II. Satuan Data

Komputer bekerja atas dasar sistem biner berupa 0 dan 1 yang disebut bit.

Bit merupakan satuan data terkecil dalam sistem komputer. Bit-bit dapat digunakan untuk menyusun karakter apa saja. Sebuah karakter dinyatakan dengan 8 bit atau 16 bit.

1. Byte

- Byte merupakan satuan yang digunakan untuk menyatakan sebuah karakter pada sistem ASCII atau EBCDIC
- 1 byte = 8 bit

2. Kilobyte (KB)

- Biasa digunakan untuk berkas gambar berukuran kecil
- 1 kilobyte = 1024 byte

3. Megabyte (MB)

- Biasa digunakan untuk menyatakan kapasitas RAM dalam PC
- 1 MB = 1024 KB = 1.048.576 byte

4. Gigabyte (GB)

- Biasa digunakan untuk menyatakan kapasitas harddisk dalam PC
- 1 GB = 1024 MB = 1.073.741.824 byte

5. Terabyte (TB)

- Biasa digunakan untuk menyatakan kapasitas *harddisk* dalam *mainframe*
- $1 \text{ TB} = 1024 \text{ GB} = 1.009.511.627.776 \text{ byte}$

6. Petabyte (PB)

- $1 \text{ PB} = 1024 \text{ TB}$

III. Sistem Pengkodean

- Sistem yang digunakan untuk mengkodekan karakter bermacam-macam.
- Data disimpan dalam memori komputer menempati posisi 1 byte, yang menggunakan kombinasi dari digit Biner.
- Komputer berbeda dalam menggunakan kode biner untuk mewakili sebuah karakter.
- Ada beberapa kode yang akan dibahas, yaitu BCD, EBCDIC, ASCII dan Unicode

1. BCD (*Binary Coded Decimal*)

- Merupakan kode biner yang digunakan hanya untuk mewakili nilai digit desimal saja.
- Sebuah karakter BCD dinyatakan dengan 4 bit
- Karakter yang tersedia sebanyak 10 angka, yaitu angka 0,1,2,3,4,5,6,7,8,9
- Digunakan pada komputer generasi pertama.

BCD 4 Bit	Digit Desimal
0000	0
0001	1
0010	2
0011	3
0100	4

BCD 4 Bit	Digit Desimal
0101	5
0110	6
0111	7
1000	8
1001	9

2. EBCDIC (*Extended Binary Coded Decimal Interchange Code*)

- EBCDIC dikembangkan oleh IBM, yang diterapkan pada berbagai komputer *mainframe*
- Sebuah karakter dinyatakan dengan 8 bit
- Karakter yang tersedia sebanyak $2^8 = 256$ karakter
- Digunakan pada komputer generasi ketiga

3. ASCII (*American Standard Code for Information Interchange*)

- ASCII dikembangkan oleh ANSI (*American National Standard Institute*)
- Sebuah karakter ASCII dinyatakan dengan 8 bit
- Karakter yang tersedia sebanyak 226 karakter, meliputi huruf, angka, dan spesial karakter, termasuk simbol Yunani dan karakter grafis

Tabel EBCDIC 8 bit

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	DS		SP	&										0
1	SOH	DC1	DOS						a	j			A	J		1
2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
3	ETX	DC3							c	l	t		C	L	T	3
4	PF	RES	BYP	PN					d	m	u		D	M	U	4
5	HT	NL	LF	RS					e	n	v		E	N	V	5
6	LC	BS	ETB	UC					f	o	w		F	O	W	6
7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
8	CAN								h	q	y		H	Q	Y	8
9	RLF	EM							i	r	z		I	R	Z	9
A	SMM	CC	SM			!		:								
B	VT					\$	'	#								
C	FF	IFS		DC4	<	"	%	@								
D	CR	IGS	ENQ	NAK	()		.								
E	SO	IRS	ACK		□	;	>	=								
F	SI	IUS	BEL	SUB			?									

Tabel ASCII 8 bit

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	À	160	A0	á	192	C0	Ì	224	E0	α
129	81	Á	161	A1	â	193	C1	Í	225	E1	β
130	82	Â	162	A2	ã	194	C2	Î	226	E2	Γ
131	83	Ã	163	A3	ä	195	C3	Ï	227	E3	Π
132	84	Ä	164	A4	å	196	C4	Ĳ	228	E4	Σ
133	85	Å	165	A5	æ	197	C5	Œ	229	E5	σ
134	86	Æ	166	A6	ç	198	C6	ſ	230	E6	τ
135	87	Ç	167	A7	è	199	C7	ı	231	E7	γ
136	88	È	168	A8	é	200	C8	Ĳ	232	E8	ϰ
137	89	É	169	A9	ê	201	C9	Œ	233	E9	Θ
138	8A	Ê	170	AA	ë	202	CA	Œ	234	EA	Ω
139	8B	Ë	171	AB	Ě	203	CB	Œ	235	EB	δ
140	8C	Ě	172	AC	ı	204	CC	Œ	236	EC	ϰ
141	8D	ı	173	AD	ı	205	CD	Œ	237	ED	ϰ
142	8E	ı	174	AE	ı	206	CE	Œ	238	EE	€
143	8F	ı	175	AF	ı	207	CF	Œ	239	EF	ı
144	90	ı	176	B0	ı	208	D0	ı	240	F0	ı
145	91	ı	177	B1	ı	209	D1	ı	241	F1	ı
146	92	ı	178	B2	ı	210	D2	ı	242	F2	ı
147	93	ı	179	B3	ı	211	D3	ı	243	F3	ı
148	94	ı	180	B4	ı	212	D4	ı	244	F4	ı
149	95	ı	181	B5	ı	213	D5	ı	245	F5	ı
150	96	ı	182	B6	ı	214	D6	ı	246	F6	ı
151	97	ı	183	B7	ı	215	D7	ı	247	F7	ı
152	98	ı	184	B8	ı	216	D8	ı	248	F8	ı
153	99	ı	185	B9	ı	217	D9	ı	249	F9	ı
154	9A	ı	186	BA	ı	218	DA	ı	250	FA	ı
155	9B	ı	187	BB	ı	219	DB	ı	251	FB	ı
156	9C	ı	188	BC	ı	220	DC	ı	252	FC	ı
157	9D	ı	189	BD	ı	221	DD	ı	253	FD	ı
158	9E	ı	190	BE	ı	222	DE	ı	254	FE	ı
159	9F	ı	191	BF	ı	223	DF	ı	255	FF	ı

4. Unicode

- Sebuah karakter Unicode dinyatakan dengan 16 bit
- Karakter yang tersedia sebanyak 65.536 karakter, meliputi huruf, angka, dan spesial karakter, termasuk simbol Yunani, karakter grafis, simbol Arab dan Cina

1. Konversi dari Bilangan Desimal ke Biner

- Dengan cara membagi bilangan desimal dengan 2 (basis biner) sampai tidak bisa dibagi lagi
- Kemudian sisa pembagian diurutkan dari bawah ke atas dalam format 8 bit
- Contoh nilai 89_{10} akan dikonversikan menjadi Biner

Konversi dari Bilangan Desimal ke Biner (Lanjutan)


$$\begin{array}{r} 89 \\ \underline{2:} \\ 44 \text{ sisa } 1 \\ \underline{2:} \\ 22 \text{ sisa } 0 \\ \underline{2:} \\ 11 \text{ sisa } 0 \\ \underline{2:} \\ 5 \text{ sisa } 1 \\ \underline{2:} \\ 2 \text{ sisa } 1 \\ \underline{2:} \\ 1 \text{ sisa } 0 \end{array}$$

Dituliskan dari bawah ke atas: **1011001**
Karena penulisan dengan 8 bit, maka
 $89_{10} = \mathbf{0101\ 1001}_2$

2. Konversi dari Bilangan Desimal ke Oktal

- Dengan cara membagi bilangan desimal dengan 8 (basis oktal) sampai tidak bisa dibagi lagi
- Cara yang digunakan sama dengan bilangan biner
- Contoh nilai 147_{10} akan dikonversikan menjadi Oktal

Konversi dari Bilangan Desimal ke Oktal (Lanjutan)

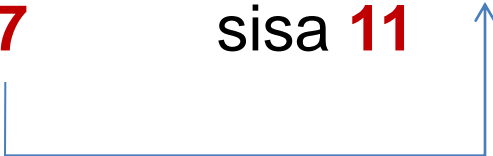
$$\begin{array}{r} 147 \\ \underline{8 :} \\ 18 \end{array} \quad \text{sisa } 3$$
$$\begin{array}{r} 18 \\ \underline{8 :} \\ 2 \end{array} \quad \text{sisa } 2$$


- Dituliskan dari bawah ke atas: **223**
- Maka hasilnya menjadi $147_{10} = 223_8$

3. Konversi dari Bilangan Desimal ke Hexadesimal

- Dengan cara membagi bilangan desimal dengan 16 (basis hexa) sampai tidak bisa dibagi lagi
- Cara yang digunakan sama dengan bilangan biner
- Contoh nilai 123_{10} akan dikonversikan menjadi Hexa

Konversi dari Bilangan Desimal ke Hexadesimal (Lanjutan)

$$\begin{array}{r} 123 \\ \underline{16 :} \\ 7 \end{array} \quad \text{sisanya } 11$$


- Sisa 11 dikodekan menjadi B
- Maka hasilnya menjadi $123_{10} = 7B_{16}$

4. Konversi dari Bilangan Biner ke Desimal

- Dengan cara mengalikan masing-masing bit biner dalam bilangan sesuai dengan *radix* dan *position value*-nya
- Contoh bit **11 0101₂** akan dikonversikan menjadi Desimal

110101₂

→	$1 \times 2^0 =$	1
→	$0 \times 2^1 =$	0
→	$1 \times 2^2 =$	4
→	$0 \times 2^3 =$	0
→	$1 \times 2^4 =$	16
→	$1 \times 2^5 =$	<u>32</u> ₊
		53

Maka hasil di samping
dituliskan:

$$\mathbf{11\ 0101_2 = 53_{10}}$$

5. Konversi dari Bilangan Biner ke Oktal

- Dengan cara membagi digit biner tersebut ke dalam tiga digit dari kanan
- Ketiga digit tersebut kemudian dikonversikan menjadi desimal
- Contoh bit **1010 1011**₂ akan dikonversikan menjadi Oktal

Biner	10	101	011
Desimal	2	5	3

Maka dituliskan menjadi **1010 1011**₂ = **253**₈

6. Konversi dari Bilangan Biner ke Hexadesimal

- Dengan cara membagi digit biner tersebut ke dalam empat digit dari kanan
- Keempat digit tersebut kemudian dikonversikan menjadi desimal
- Contoh bit **10101011**₂ akan dikonversikan menjadi Hexa

Biner	1010	1011
Desimal	10	11
Hexa	A	B

Maka dituliskan menjadi **1010 1011**₂ = **AB**₁₆

7. Konversi dari Bilangan Oktal ke Desimal

- Dengan cara mengalikan masing-masing bit oktal dalam bilangan sesuai dengan *radix* dan *position value*-nya
- Contoh bit **371₈** akan dikonversikan menjadi Desimal

371₈

1	$\times 8^0 =$	1
7	$\times 8^1 =$	56
3	$\times 8^2 =$	<u>192</u> ₊
		249

Maka hasil disamping dituliskan:

$$\mathbf{371_8 = 249_{10}}$$

8. Konversi dari Bilangan Oktal ke Biner

- Dengan cara mengkonversikan setiap satu digit oktal menjadi tiga digit biner
- Contoh bit 71_8 akan dikonversikan menjadi Biner

Oktal	7	1
Biner	111	001

Maka dituliskan menjadi $71_8 = 0011\ 1001_2$

9. Konversi dari Bilangan Oktal ke Hexadesimal

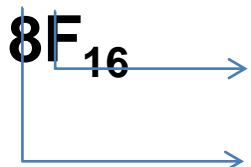
- Konversi ini tidak dapat dilakukan secara langsung, tetapi harus dikonversikan terlebih dahulu ke Desimal atau Biner
- Contoh bit 243_8 akan dikonversikan menjadi Hexa

Oktal	2	4	3
Biner	010	100	011
	1010		0011
Hexa	A		3

Maka dituliskan menjadi $243_8 = A3_{16}$

10. Konversi dari Bilangan Hexadesimal ke Desimal

- Dengan cara mengalikan masing-masing bit hexa dalam bilangan sesuai dengan *radix* dan *position value*-nya
- Contoh bit $8F_{16}$ akan dikonversikan menjadi Desimal


$$\begin{array}{rcl} F \times 16^0 & = & 15 \\ 8 \times 16^1 & = & \underline{128} \\ & & 143 \end{array}$$

Maka hasil disamping dituliskan:

$$8F_{16} = 143_{10}$$

11. Konversi dari Bilangan Hexadesimal ke Biner

- Dengan cara mengkonversikan setiap satu digit hexa menjadi empat digit biner
- Contoh bit $8F_{16}$ akan dikonversikan menjadi Biner

Hexa	8	F
Biner	1000	1111

Maka dituliskan menjadi $8F_{16} = 1000\ 1111_2$

12. Konversi dari Bilangan Hexadesimal ke Oktal

- Konversi ini tidak dapat dilakukan secara langsung, tetapi harus dikonversikan terlebih dahulu ke Desimal atau Biner sama dengan konversi dari oktal ke hexa
- Contoh bit $8F_{16}$ akan dikonversikan menjadi Oktal

Hexa	8		F	
Biner	1000		1111	
	10	001	111	
Oktal	2	1	7	

Maka dituliskan menjadi $8F_{16} = 217_8$

LATIHAN

Dosen diharapkan memberikan contoh dan latihan konversi bilangan

Pertemuan 3

ARRAY DIMENSI BANYAK

ARRAY DIMENSI TIGA (Three Dimensional Array)

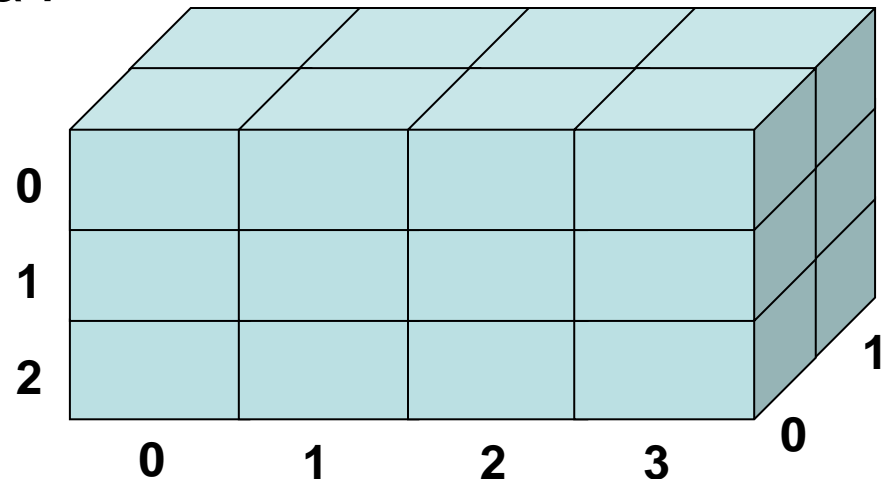
Digunakan untuk mengelola data dalam bentuk 3 dimensi atau tiga sisi.

Deklarasi :

Nama_Array[index1] [index2] [index3];

Misal : A [3][4][2];

Penggambaran secara Logika :



Menentukan jumlah elemen dalam Array dimensi 3 :

n

π (index array)

$i=1$

π = Perkalian dari statemen sebelumnya

Contoh :

Suatu Array X dideklarasikan sbb :

A [3][4][2]; maka jumlah elemen Array dimensi tiga tersebut adalah :

$$(3) * (4) * (2) = 24$$

PEMETAAN (MAPPING) ARRAY DIMENSI TIGA KE STORAGE

Rumus :

$$\text{@M[m][n][p]} = \text{M[0][0][0]} + \{((m-1) * (\text{jum.elemen2} * \text{jum.elemen3})) + ((n-1) * (\text{jum.elemen3})) + ((p-1)) * L\}$$

Contoh :

Suatu Array A dideklarasikan sebagai berikut :

int A [2][4][3], dengan alamat awal index A[0][0][0] berada di 0011_(H) dan ukuran type data int = 2 Tentukan berapa alamat array di A[2][3][2] ?

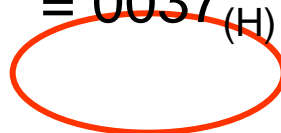
Contoh Pemetaan :

Penyelesaian :

$$1. \text{Tentukan jumlah elemen array A [2][4][3]} \\ = (2) * (4) * (3) = 24$$

$$2. @M[m][n][p] = M[0][0][0] + \{((m-1) * (\text{jum.elemen2} * \text{jum.elemen3})) + ((n-1) * (\text{jum.elemen3})) + ((p-1))\} * L$$

$$\begin{aligned} A[2][3][2] &= 0011_{(H)} + \{((2-1) * 4 * 3) + ((3-1) * 3) + (2-1)\} * 2 \\ &= 0011_{(H)} + \{12 + 6 + 1\} * 2 \\ &= 0011_{(H)} + 38_{(D)} \quad 26_{(H)} \\ &= 0011_{(H)} + 26_{(H)} \xrightarrow{\text{orange arrow}} \\ &= 0037_{(H)} \end{aligned}$$



Contoh Program Array Dimensi 3

```
1 import numpy as np
2
3 arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
4
5 print(arr[0, 1, 2])
6
```

Ln: 5, Col: 20

 Run

 Share

Command Line Arguments



6

Buatlah program array multi dimensi dengan Python

TRINGULAR ARRAY (ARRAY SEGITIGA)

Tringular Array dapat merupakan Upper Tringular (seluruh elemen di bawah diagonal utama = 0), ataupun Lower Tringular (seluruh elemen di atas diagonal utama = 0).

Dalam Array Lower Tringular dengan N baris, jumlah maksimum elemen ≤ 0 pada baris ke- I adalah $= I$, karenanya total elemen ≤ 0 , tidak lebih dari

$$\sum_{I=1}^N I = N(N+1) / 2$$

Contoh Tringular Array

X	X	X	X	X	X	X	0	0	0	0	0
0	X	X	X	X	X	X	X	0	0	0	0
0	0	X	X	X	X	X	X	X	0	0	0
0	0	0	X	X	X	X	X	X	X	0	0
0	0	0	0	X	X	X	X	X	X	X	0
0	0	0	0	0	X	X	X	X	X	X	X

(a)

(b)

Gambar (a) Upper Triangular Array
(b) Lower Triangular Array

Tringular Array (Lanjutan)

Contoh :

Diketahui suatu array segitiga atas memiliki 3 baris dan kolom, tentukan berapakah jumlah elemen yang bukan nol pada array tersebut.

$$\begin{aligned} I &= N(N+1) / 2 \\ I &= 3 (3+1) / 2 \\ &= 12 / 2 \\ &= 6 \end{aligned}$$

Contoh bentuk array nya adalah seperti dibawah ini :

10	20	30		5	10	15
0	40	50		0	20	25
0	0	60		0	0	30

Dan lain-lain

Triangular Array (Lanjutan)

Suatu Array Upper Triangular dan Array Lower Triangular dapat dengan order yang sama, dapat disimpan sebagai suatu array dengan order yang berbeda, Contohnya :

$$A = \begin{vmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{vmatrix} \quad B = \begin{vmatrix} 7 & 0 & 0 \\ 8 & 9 & 0 \\ 11 & 12 & 13 \end{vmatrix}$$

dapat disimpan sebagai Array C berorder (3 X 4)

$$\begin{vmatrix} 7 & 1 & 2 & 3 \\ 8 & 9 & 4 & 5 \\ 11 & 12 & 13 & 6 \end{vmatrix}$$

SPARSE ARRAY (ARRAY JARANG)

Suatu Array yang sangat banyak elemen nol-nya, contohnya adalah Array A pada Gambar berikut :

0	0	0	0	1	0	0	2
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	4	0	0	0	0
0	0	0	0	0	0	0	2

Latihan

1. Suatu array A dideklarasikan sbb:

Float $A[5][5][5]$ dengan alamat awal $A[0][0][0] = 0021(H)$, berapakah alamat array $A[2][3][2]$ dan $A[5][4][3]$?

2. Suatu array B dideklarasikan sbb:

Char $B[3][4][3]$ dengan alamat awal $A[0][0][0] = 0021(H)$, berapakah alamat array $A[2][3][2]$ dan $A[3][4][3]$?

Pertemuan 4

LINKED LIST

BAGAIMANA LINKED LIST MENYIMPAN DATA

Linked List, di sisi lain, berbeda dengan array. Linked List, tidak menyimpan data di lokasi memori yang berdekatan. Untuk setiap item di lokasi memori, Linked List menyimpan nilai item dan referensi atau penunjuk ke item berikutnya. Sepasang item Linked List dan referensi ke item berikutnya merupakan sebuah node.

Misal, jika sebuah node terdiri dari 34 | 10, artinya nilai node tersebut adalah 30, sedangkan item berikutnya disimpan di lokasi memori "10". Untuk melintasi Linked List, Anda hanya perlu mengetahui lokasi memori atau referensi dari node pertama, node lainnya dapat dilintasi secara berurutan menggunakan referensi ke elemen berikutnya di setiap node.

Referensi ke node pertama juga dikenal sebagai node awal.

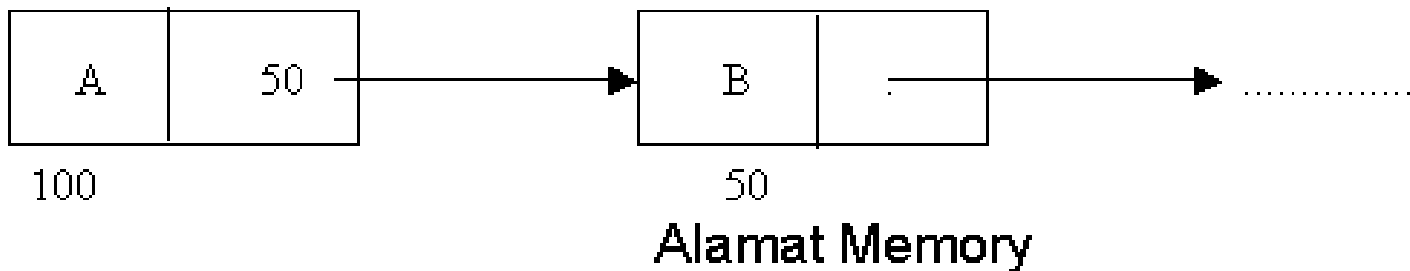
LINKED LIST vs ARRAY

- Linked List adalah struktur data dinamis yang berarti bahwa memori yang dicadangkan untuk Linked List dapat ditingkatkan atau dikurangi saat runtime. Tidak ada memori yang dialokasikan untuk struktur data Linked List sebelumnya. Setiap kali item baru diperlukan untuk ditambahkan ke link, memori untuk node baru dibuat pada saat berjalan. Di sisi lain, dalam kasus larik, memori harus dialokasikan terlebih dahulu untuk sejumlah item tertentu. Dalam kasus di mana item yang cukup tidak tersedia untuk mengisi semua indeks larik, ruang memori akan terbuang.
- Karena array membutuhkan lokasi memori yang berdekatan, sangat sulit untuk menghapus atau menyisipkan item dalam array karena lokasi memori dari sejumlah besar item harus diperbarui. Di sisi lain, item daftar tertaut tidak disimpan di lokasi memori yang berdekatan, oleh karena itu Anda dapat dengan mudah memperbarui Linked List .
- Karena fleksibilitasnya, Linked List lebih cocok untuk mengimplementasikan struktur data seperti tumpukan, antrian, dan daftar.

LINKED LIST

Salah satu Struktur Data Dinamis yang paling sederhana adalah Linked List atau Struktur Berkait atau Senarai Berantai, *yaitu suatu kumpulan komponen yang disusun secara berurutan dengan bantuan Pointer.*

Linked List (Senarai Berantai) disebut juga dengan Senarai Satu Arah (One-Way List). Masing-masing komponen dinamakan dengan Simpul (Node).

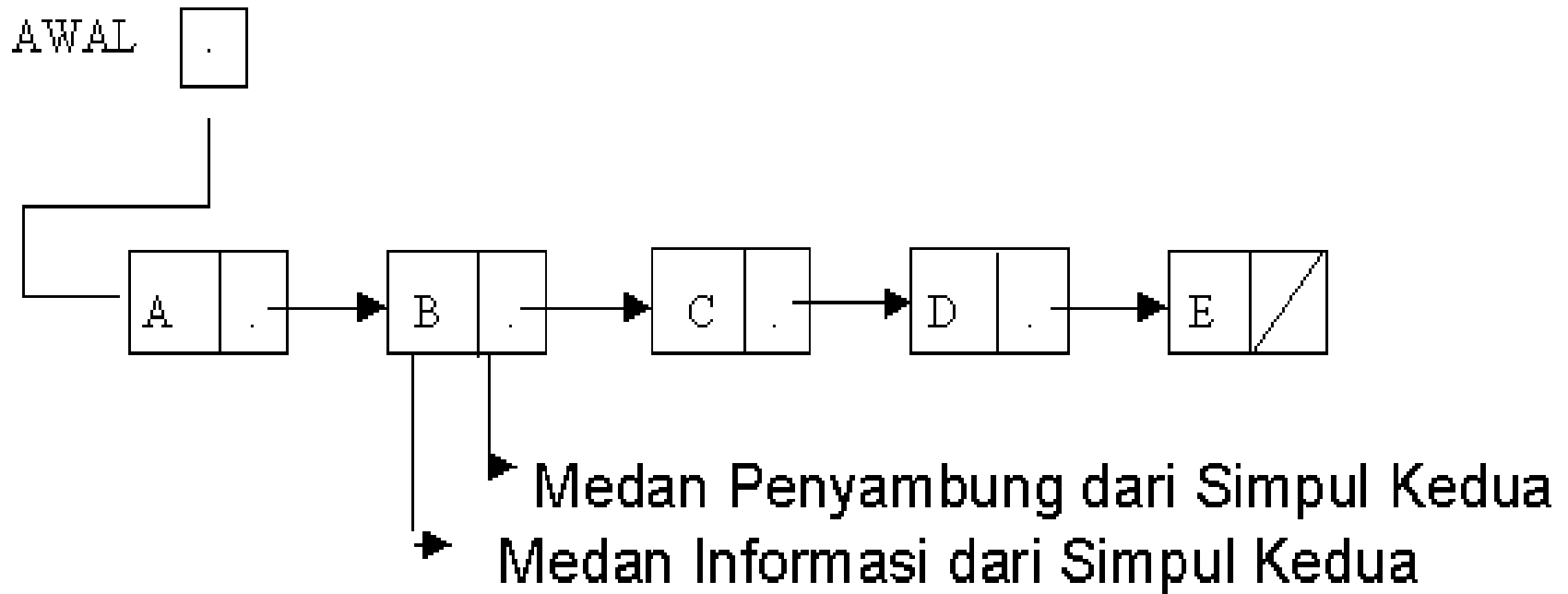


Setiap simpul dalam suatu Linked List terbagi menjadi dua bagian,yaitu :

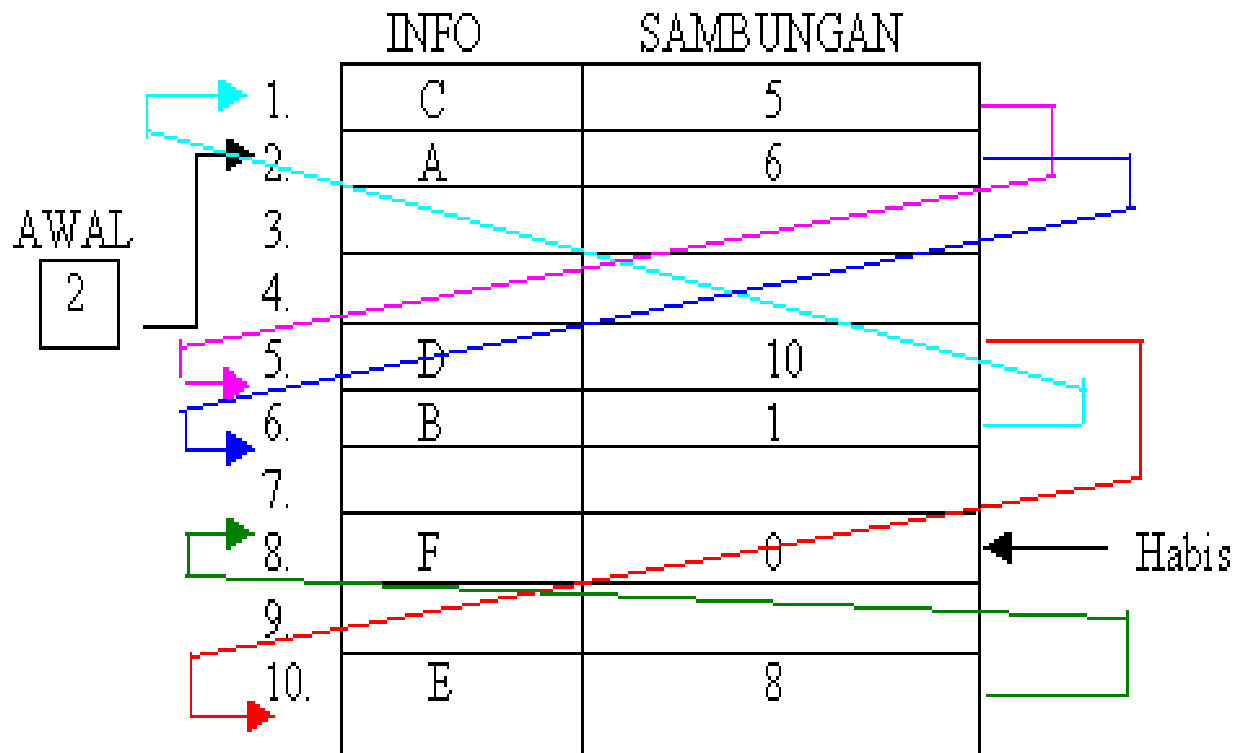
1. Medan Informasi
Berisi informasi yang akan disimpan dan diolah.
2. Medan Penyambung (Link Field)
Berisi alamat berikutnya. Bernilai 0, Jika Link tersebut tidak menunjuk ke Data (Simpul) lainnya. Penunjuk ini disebut Penunjuk Nol.

LINKED LIST (Lanjutan)

Linked juga mengandung sebuah variabel penunjuk List, yang biasanya diberi nama START (AWAL), yang berisi alamat dari simpul pertama dalam List.



Penyajian Linked List dalam Memory



Penyajian Linked List dalam Memory (Lanjutan)

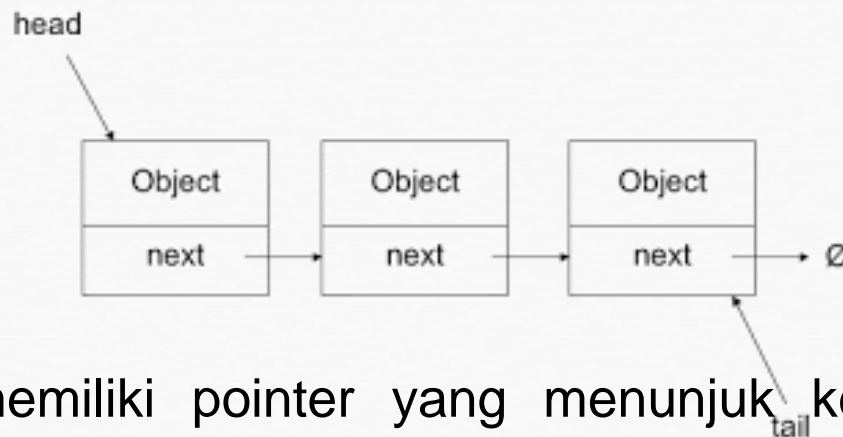
Keterangan :

AWAL	= 2	, Maka	INFO[2]	= 'A'
SAMBUNGAN[2]	= 6	, Maka	INFO[6]	= 'B'
SAMBUNGAN[6]	= 1	, Maka	INFO[1]	= 'C'
SAMBUNGAN[1]	= 5	, Maka	INFO[5]	= 'D'
SAMBUNGAN[5]	= 10	, Maka	INFO[10]	= 'E'
SAMBUNGAN[10]	= 8	, Maka	INFO[8]	= 'F'
SAMBUNGAN[8]	= 0	, Maka	Akhir Linked List	

Dari contoh diatas diperoleh untai **'ABCDEF'**

SINGLE LINKED LIST

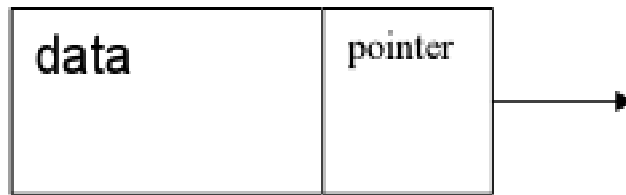
Single linked list adalah yang paling sederhana dari semua varian linked list. Setiap node dalam satu linked list berisi item dan referensi ke item berikutnya dan hanya itu. Di bagian ini, kita akan melihat cara membuat node untuk singled linked list bersama dengan fungsi untuk berbagai jenis penyisipan, traversal, dan penghapusan.



Setiap node memiliki pointer yang menunjuk ke simpul berikutnya sehingga terbentuk satu untaian, dengan demikian hanya diperlukan sebuah variabel pointer. Susunan berupa untaian semacam ini disebut **Single Linked List**. Biasanya Linked List pada node terakhir akan menunjuk ke NULL, dimana NULL memiliki nilai khusus yang artinya tidak menunjuk ke mana-mana.

Bentuk Node Single Linked List

- Single : field pointer-nya hanya satu dan satu arah, pada akhir node pointernya menunjuk NULL



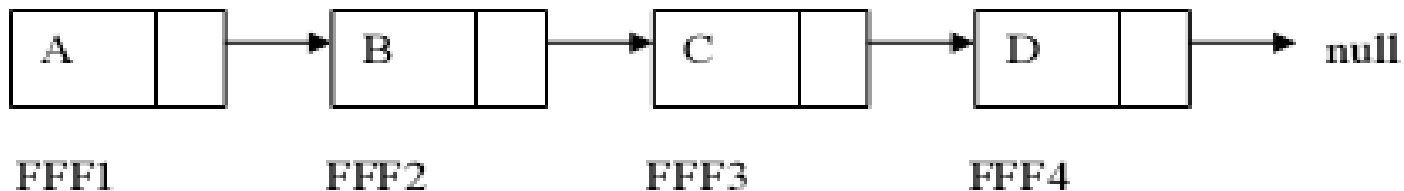
Menempati alamat memori tertentu

- Linked List : node-node tersebut saling terhubung satu sama lain.

Single Linked List (Lanjutan)

Setiap node pada linked list mempunyai field yang berisi pointer ke node berikutnya, dan juga memiliki field yang berisi data.

Node terakhir akan menunjuk ke NULL yang akan digunakan sebagai kondisi berhenti pada saat pembacaan isi linked list.



Pembuatan Single Linked List

Deklarasi Node :

```
class Node:  
def __init__(self, data):  
self.item = data  
self.ref = None
```

Keterangan:

- class node kami akan berisi dua variabel item dan ref. Nilai item akan ditetapkan oleh nilai yang diteruskan melalui konstruktor, sedangkan referensi awalnya akan disetel ke nol.

Pembuatan Class pada Single Linked List

Selanjutnya, kita perlu membuat class untuk linked list. Kelas ini akan berisi metode untuk menyisipkan, menghapus, melintasi, dan mengurutkan daftar. Awalnya, kelas hanya akan berisi satu anggota `start_node` yang akan menunjuk ke simpul awal atau pertama dari daftar. Nilai `start_node` akan disetel ke nol menggunakan konstruktor karena daftar tertaut akan kosong pada saat pembuatan. Skrip berikut membuat kelas untuk daftar tertaut.

```
class LinkedList:
```

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

```
        self.start_node = None
```

Pembuatan Class pada Single Linked List (lanjutan)

- Selanjutnya, kita perlu membuat class untuk linked list. Kelas ini akan berisi metode untuk menyisipkan, menghapus, melintasi, dan mengurutkan daftar. Awalnya, kelas hanya akan berisi satu anggota `start_node` yang akan menunjuk ke simpul awal atau pertama dari daftar. Nilai `start_node` akan disetel ke nol menggunakan konstruktor karena daftar tertaut akan kosong pada saat pembuatan. Skrip berikut membuat kelas untuk daftar tertaut.

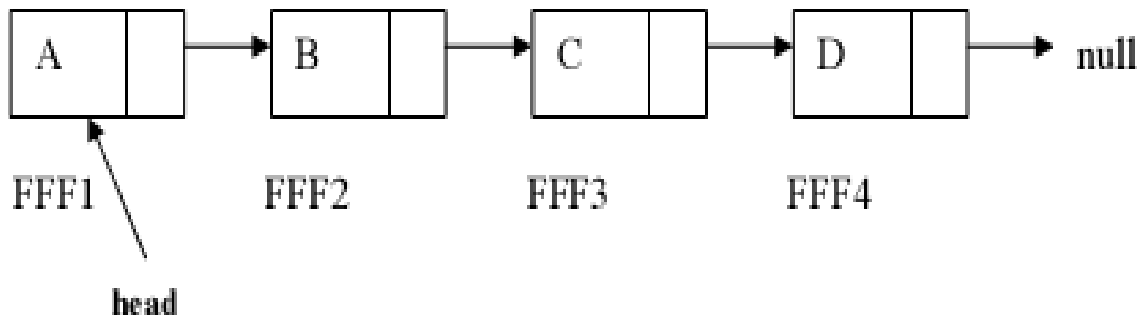
class LinkedList:

def __init__(self):

self.start_node = None

Menambahkan item pada Singled Linked List di Awal

- Tergantung pada lokasi tempat Anda ingin menyisipkan item, ada berbagai cara untuk menyisipkan item dalam singled .



- Cara paling sederhana untuk menyisipkan item dalam satu linked list adalah dengan menambahkan item di awal. Fungsi berikut menyisipkan item di awal linked list. Tambahkan fungsi ini ke kelas LinkedList yang kita buat sebelumnya.

Deklarasi Penambahan di Awal pada Single Linked List

```
def insert_at_start(self, data):  
    new_node = Node(data)  
    new_node.ref=self.start_node  
    self.start_node= new_node
```

Pada skrip di atas, kita membuat metode `insert_at_start ()`, metode tersebut menerima satu parameter, yang pada dasarnya adalah nilai item yang ingin kita masukkan. Di dalam metode ini, kita cukup membuat objek dari kelas `Node` dan menetapkan referensinya ke `start_node` karena `start_node` sebelumnya menyimpan simpul pertama, yang setelah penyisipan simpul baru di awal akan menjadi simpul kedua. Oleh karena itu, kami menambahkan referensi `start_node` ke variabel `ref` dari node baru. Sekarang karena `new_node` adalah node pertama, kami menetapkan nilai variabel `start_node` ke `new_node`.

Menambahkan Item pada Singled Linked List di Akhir

Memasukkan Item di Akhir Fungsi berikut digunakan untuk menambahkan item di akhir linked list.

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

Menambahkan Item pada Singled Linked List di Akhir

Dalam skrip di atas, membuat fungsi `insert_at_end ()`, yang menyisipkan elemen di akhir linked list. Nilai item yang ingin disisipkan sebagai argumen ke fungsi. Fungsinya terdiri dari dua bagian. Pertama kita periksa apakah linked list kosong atau tidak, jika linked list kosong, yang harus kita lakukan hanyalah mengatur nilai variabel `start_node` ke objek `new_node`. Di sisi lain, jika list sudah berisi beberapa node. Inisialisasi variabel `n` dengan simpul awal. Kemudian melakukan iterasi melalui semua node dalam list menggunakan loop sementara. Perulangan berakhir saat mencapai simpul terakhir. Kemudian menetapkan referensi node terakhir ke `new_node` yang baru dibuat. Tambahkan fungsi `insert_at_end ()` ke kelas `LinkedList`.

Menambahkan Item Setelah Item Lainnya

Menambahkan item setelah item lain dalam singled linked list. Untuk melakukannya, kita bisa menggunakan fungsi `insert_after_item ()` seperti yang didefinisikan di bawah ini:

```
def insert_after_item(self, x, data):  
    n = self.start_node  
    print(n.ref)  
    while n is not None:  
        if n.item == x:  
            break  
        n = n.ref  
    if n is None:  
        print("item not in the list")  
    else:  
        new_node = Node(data)  
        new_node.ref = n.ref  
        n.ref = new_node
```

Menambahkan Item Setelah Item Lainnya (Lanjutan)

Fungsi `insert_after_item ()` menerima dua parameter: `x` dan `data`. Parameter pertama adalah item setelah itu Anda ingin memasukkan node baru sedangkan parameter kedua berisi nilai untuk node baru. Mulai dengan membuat variabel baru `n` dan menugaskan variabel `start_node` untuk itu. Selanjutnya, menelusuri linked list menggunakan while loop. Perulangan while dijalankan hingga `n` menjadi `None`. Selama iterasi, memeriksa apakah nilai yang disimpan di node saat ini sama dengan nilai yang dilewatkan oleh parameter `x`. Jika perbandingan mengembalikan nilai `true`, maka perulangan dihentikan. Selanjutnya, jika item ditemukan, variabel `n` tidak akan menjadi `None`. Referensi `new_node` disetel ke referensi yang disimpan oleh `n` dan referensi `n` disetel ke `kode_baru`. Tambahkan fungsi `insert_after_item ()` ke kelas `LinkesList`.

Menambahkan Item Setelah Item Lainnya (Lanjutan)

```
def insert_before_item(self, x, data):  
    if self.start_node is None:  
        print("List has no element")  
        return  
    if x == self.start_node.item:  
        new_node = Node(data)  
        new_node.ref = self.start_node  
        self.start_node = new_node  
        return  
    n = self.start_node  
    while n.ref is not None:  
        if n.ref.item == x:  
            break  
        n = n.ref  
    if n.ref is None:  
        print("item not in the list")  
    else:  
        new_node = Node(data)  
        new_node.ref = n.ref  
        n.ref = new_node
```

Menambahkan Linked List

Meskipun kita dapat menambahkan item satu per satu menggunakan salah satu fungsi penyisipan. Mari buat fungsi yang meminta pengguna untuk memasukkan jumlah elemen di node dan kemudian elemen individu dan memasukkan elemen itu ke dalam linked list.

```
def make_new_list(self):  
    nums = int(input("How many nodes do you want to create: "))  
    if nums == 0:  
        return  
    for i in range(nums):  
        value = int(input("Enter the value for the node:"))  
        self.insert_at_end(value)
```

Menambahkan Linked List (Lanjutan)

Dalam skrip di atas, fungsi `make_new_list ()` pertama-tama meminta pengguna untuk jumlah item dalam daftar. Selanjutnya menggunakan for-loop, pengguna diminta untuk memasukkan nilai untuk setiap node, yang kemudian dimasukkan ke dalam daftar tertaut menggunakan fungsi `insert_at_end ()`.

Menghapus Element di Awal

Menghapus elemen atau item dari awal daftar tertaut sangatlah mudah. Kita harus mengatur referensi `start_node` ke node kedua yang dapat kita lakukan hanya dengan menetapkan nilai referensi dari node awal (yang menunjuk ke node kedua) ke node awal seperti yang ditunjukkan di bawah ini:

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

Pada skrip di atas, periksa dulu apakah daftar itu kosong atau tidak. Jika daftar kosong tampilkan pesan bahwa daftar tidak memiliki elemen untuk dihapus. Jika tidak, tetapkan nilai `start_node.ref` ke `start_node` tersebut. `Start_node` sekarang akan mengarah ke elemen kedua. Tambahkan fungsi `delete_at_start ()` ke class `LinkedList`.

Menghapus Element di Akhir

Untuk menghapus elemen dari akhir daftar, kita hanya perlu mengulang melalui linked list sampai elemen terakhir kedua, dan kemudian perlu mengatur referensi elemen terakhir kedua ke none, yang akan mengubah elemen terakhir kedua menjadi elemen terakhir. Script untuk fungsi `delete_at_end` adalah sebagai berikut:

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

Menghapus Berdasarkan Nilai Element

Untuk menghapus elemen berdasarkan nilai, pertama-tama kita harus menemukan node yang berisi item dengan nilai yang ditentukan dan kemudian menghapus node tersebut. Menemukan item dengan nilai yang ditentukan sangat mirip dengan pencarian item. Setelah item yang akan dihapus ditemukan, referensi node sebelum item diatur ke node yang ada setelah item dihapus.

```
def delete_element_by_value(self, x):  
    if self.start_node is None:  
        print("The list has no element to delete")  
        return  
  
    # Deleting first node  
    if self.start_node.item == x:  
        self.start_node = self.start_node.ref  
        return  
        n = self.start_node  
    while n.ref is not None  
    if n.ref.item == x:  
        break  
        n = n.ref  
    if n.ref is None:  
        print("item not found in the list")  
    else:  
        n.ref = n.ref.ref
```

Contoh program single linked list

```
class node:
```

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

```
        self.data = initdata
```

```
        self.next = None
```

```
    def getdata (self):
```

```
        return self.data
```

```
    def getnext(self):
```

```
        return self.next
```

```
    def setdata (self,newdata):
```

```
        self.data = newdata
```

```
    def setnext (self,newnext):
```

```
        self.next = newnext
```

```
class orderedlist:
```

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

```
        self.head = None
```

```
    def show (self):
```

```
        current = self.head
```

```
        print("Head -> ", end = "")
```

```
        while current != None:
```

```
            print(current.getdata(), end="-> ")
```

```
            current = current.getnext()
```

```
        print("None")
```

```
    def isempty(self):
```

```
        return self.head == None
```

```
    def add(self,item):
```

```
        temp = node(item)
```

```
        temp.setnext(self.head)
```

```
        self.head = temp
```

```
    def size(self):
```

```
        current = self.head
```

```
        count = 0
```

```
        while current != None :
```

```
count += 1
```

```
current = current.getnext()
```

```
return count
```

```
def search(self):
```

```
    current = self.head
```

```
    s = int(input("masukkan data yang akan dicari : "))
```

```
    found = False
```

```
    while current != None and not found:
```

```
        if current.getdata() == s:
```

```
            found = True
```

```
        else:
```

```
            current = current.getnext()
```

```
    if found == True :
```

```
        print("Data ditemukan")
```

```
    else :
```

```
        print("Data tidak ditemukan")
```



```
def remove (self):
```

```
    current = self.head
```

```
    s = int(input("masukkan data yang akan dihapus : "))
```

```
    previous = None
```

```
    found = False
```

```
    while not found and current != None:
```

```

if current.getdata() == s:
    print("Data yang akan dihapus tidak ditemukan")
    found = True
else:
    previous = current
    current = current.getnext()
if found == True :
    if previous == None:
        self.head = current.getnext()
    else:
        previous.setnext(current.getnext())
    print("Data sudah dihapus")
else :
    print("Data yang akan dihapus tidak ditemukan")

def ganjil(self,item):
    for i in range(len(item)):
        if item[i] % 2 == 1:
            ganjil.append(item[i])
    ol.sort(ganjil)
    print(ganjil)

def genap(self,item):
    for i in range(len(item)):
        if item[i] % 2 == 0:
            genap.append(item[i])
    ol.sort(genap)
    print(genap)

def sort(self,item):
    for x in range(len(item)-1, 0, -1):
        for y in range(x):
            if item[y] > item[y+1]:
                item[y], item[y+1] = item[y+1], item[y]

def tambah(self,item):
    pil = "y"

```

```
while pil == "y":
```

```
    tam = int(input("masukkan data : "))
```

```
    lis.append(tam)
```

```
    pil = str(input("apakah anda mau memasukkan data lagi ? (y/n)"))
```

```
ol.ganjil(lis)
```

```
ol.genap(lis)
```

```
genap.reverse()
```

```
merge = ganjil + genap
```

```
for i in merge:
```

```
    ol.add(i)
```

```
ol = orderedlist()
```

```
lis =
```

```
[170411100001,170411100002,170411100003,170411100004,170411100005,170411100006]
```

```
genap=[]
```

```
ganjil=[]
```

```
merge = ganjil + genap
```



```
ol.tambah(ol)
```

```
ol.show()
```

```
print("panjang datanya adalah",orderedlist.size(ol))
```

```
orderedlist.search(ol)
```

```
orderedlist.remove(ol)
```

```
ol.show()
```

Latihan

(Dikumpulkan pada pertemuan selanjutnya)

Buatlah Ilustrasi / Penggambaran untuk menambah dan menghapus node di posisi tengah pada :

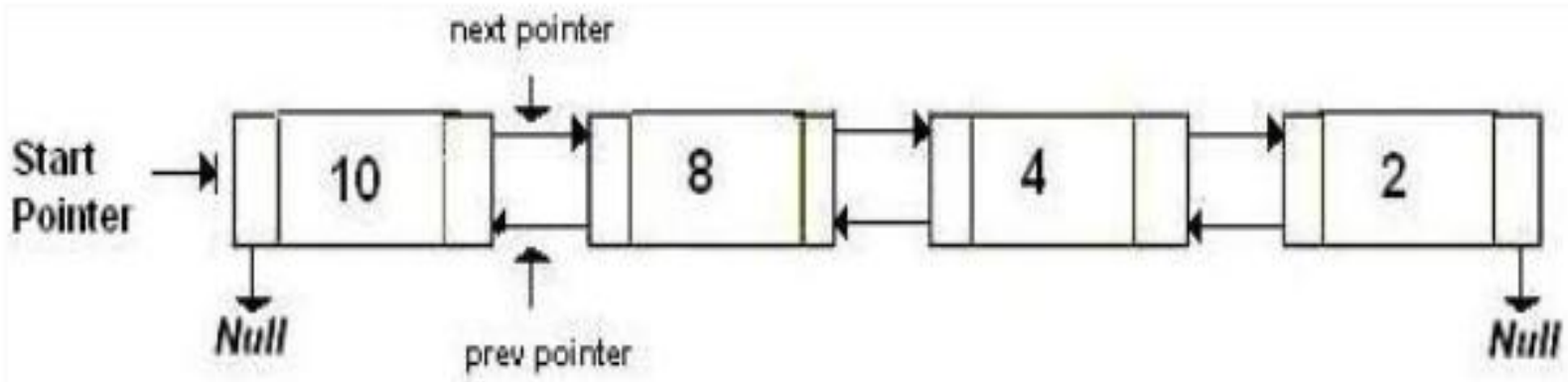
1. Single Linked List dengan Head
2. Single Linked List dengan Head & Trail

Pertemuan 5

LINKED LIST LANJUTAN

Doubly Linked List

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



Pembuatan Doubly Linked List

Membuat class untuk node

class Node:

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

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

class DoublyLinkedList:

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

Menambahkan Item pada Empty List

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

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

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

Menambahkan Item di Awal Doubly Linked List

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

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

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

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

```
        self.start_node = new_node
```

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

```
    return
```

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

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

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

```
        self.start_node = new_node
```

Menambahkan Item di Akhir Doubly Linked List

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

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

Menambahkan Item pada After Another Item

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

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


Menambahkan Item Before Another Item

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

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

Menambahkan Item Before Another Item (Lanjutan)

```
def insert_before_item(self, x, data):
```

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

```
        print("List is empty")
```

```
        return
```

```
    else:
```

```
        n = self.start_node
```

```
        while n is not None:
```

```
            if n.item == x:
```

```
                break
```

```
        n = n.nref if n is None:
```

```
            print("item not in the list")
```

```
        else:
```

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

```
            new_node.nref = n
```

```
            new_node.pref = n.pref
```

```
            if n.pref is not None:
```

```
                n.pref.nref = new_node
```

```
            n.pref = new_node
```

Menghapus Elemen dari Doubly Linked List

Menghapus Elemen di node awal

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

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

Menghapus Elemen di Node Akhir

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

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

Menghapus Elemen Berdasarkan Nilai

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

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

Implementasi Doubly Linked List

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

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

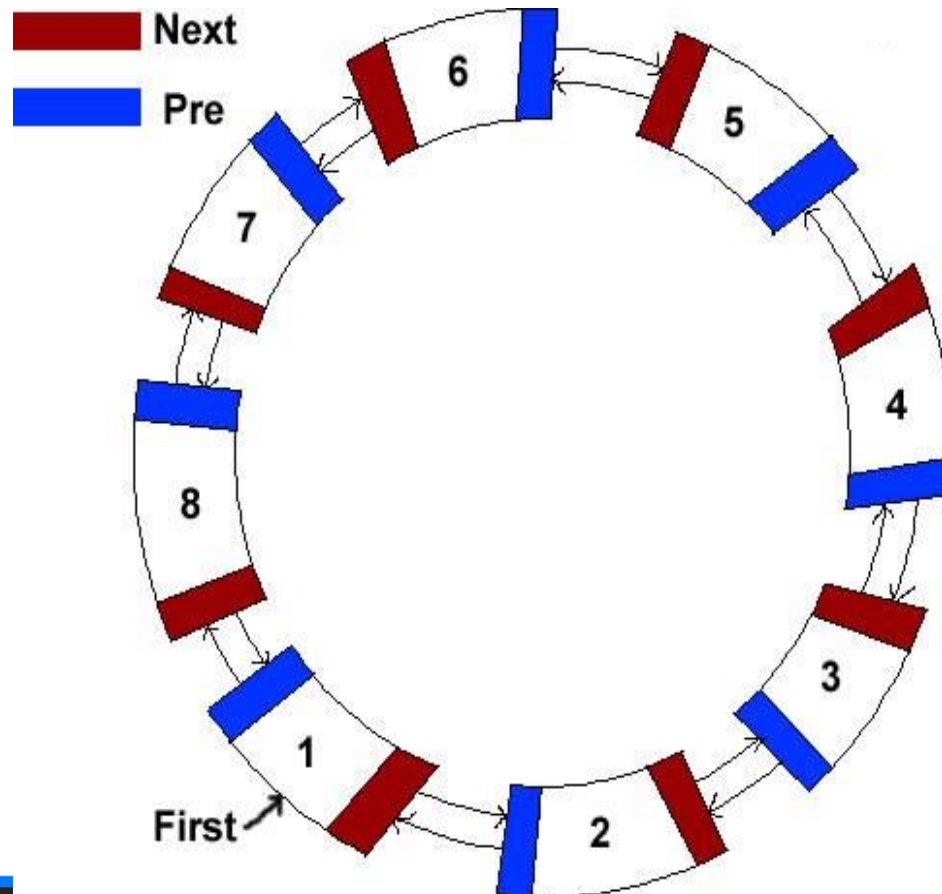
Implementasi Doubly Linked List (Lanjutan)

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

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

Circular Linked List

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



Circular Linked List

```
class Node:
```

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

```
self.data = data
```

```
self.next = None
```

```
class CircularLinkedList:
```

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

```
self.head = None
```

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

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

```
cur = self.head
```

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

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

```
new_node.next = new_node
```

```
else:
```

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

```
cur = cur.next
```

```
cur.next = new_node
```

```
self.head = new_node
```

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

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

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

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

```
else:
```

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

```
cur = self.head
```

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

```
cur = cur.next
```

```
cur.next = new_node
```

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

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

```
cur = self.head
```

```
while cur:
```

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

```
cur = cur.next
```

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

```
break
```

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

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

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

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

```
list = CircularLinkedList()
```

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

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

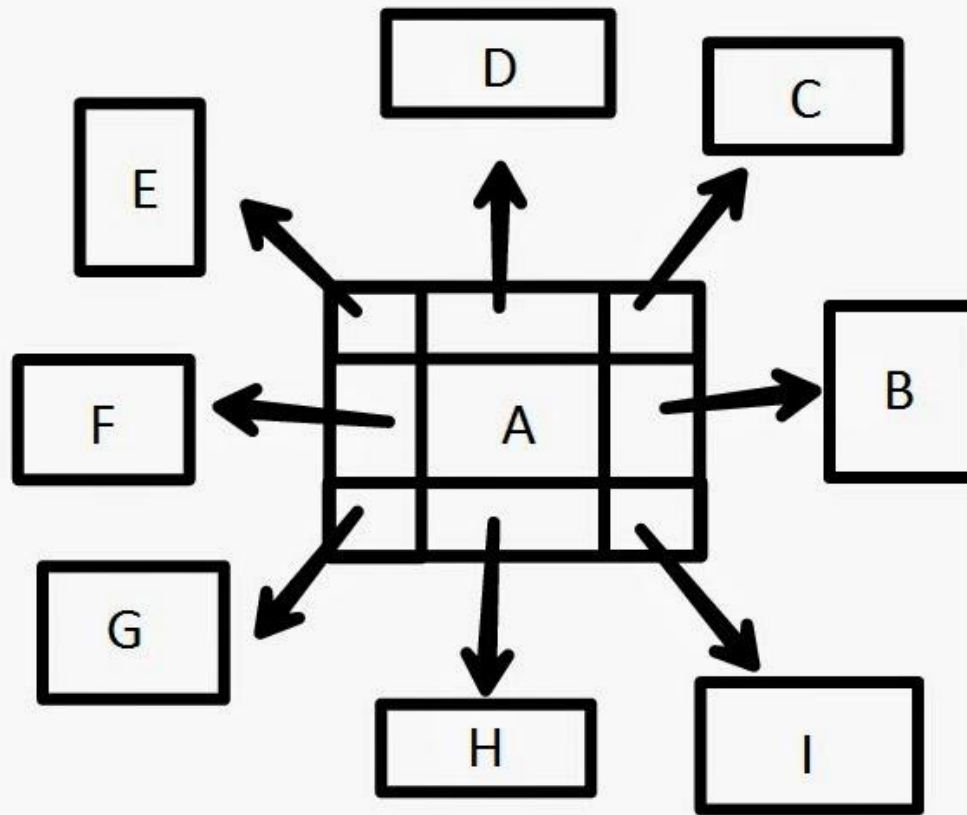
```
list.prepend(f1)
```

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

```
list.print_list()
```

Multiple Linked List

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



Pertemuan 6

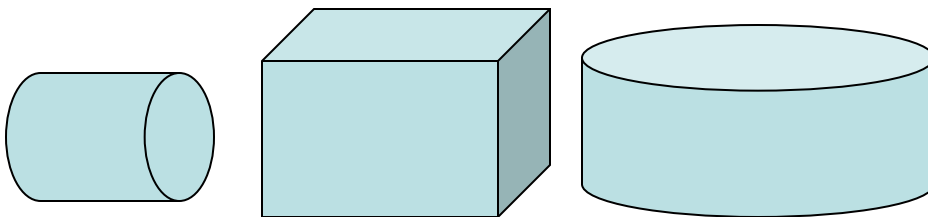
STACK atau TUMPUKAN



STACK (TUMPUKAN)

Merupakan bentuk khusus dari Linier List yang pemasukan dan penghapusan elemennya hanya dapat dilakukan pada satu posisi, yaitu posisi akhir dari List (Top)

Prinsip Stack adalah ***LAST-IN-FIRST-OUT (LIFO)***.



Klik untuk
Ilustrasi Stack

OPERASI STACK

- **EMPTY**
Untuk memeriksa apakah stack kosong
- **SIZE**
mengembalikan nilai stack
- **PUSH**
Untuk menambahkan item pada posisi paling atas
- **POP**
Untuk menghapus item paling atas (TOP)
- **TOP**
mengembalikan posisi ke elemen paling atas
- **NOEL**
menampilkan jumlah elemen pada stack

Inisialisasi

```
stack = []

def push(value):
    stack.append(value)

def pop():
    stack.pop()

def noel():
    print len(stack)

def top():
    top = len(stack) - 1
    if top < 0:
        print "Tidak terdefinisi"
    else:
        print stack[top]

def isempty():
    if len(stack) == 0:
        print "True"
    else:
        print "False"

def tampilkan(stack):
    print stack

while True:
    value = input("--> ")
```

stack = [] ==> Digunakan untuk membuat stack dengan nama "stack" yang masih kosong

def push(value): ==> Digunakan untuk mendefinisikan sebuah fungsi "push" yang diisi dengan nilai

stack.append(value) ==> Digunakan untuk menambahkan elemen pada puncak stack dengan nilai dari fungsi "push"

def pop(): ==> Digunakan untuk mendefinisikan sebuah fungsi "pop"

stack.pop() ==> Digunakan untuk menghapus elemen dari puncak stack dari fungsi "pop"

def noel(): ==> Digunakan untuk mendefinisikan sebuah fungsi "noel"

print len(stack) ==> Digunakan untuk menampilkan jumlah elemen di dalam stack dengan memanggil nama "stack" yang telah dibuat

def top(): ==> Digunakan untuk mendefinisikan sebuah fungsi "top"

top = len(stack) - 1 ==> Sebagai perhitungan untuk mengetahui elemen pada puncak stack

print stack[top] ==> Maka akan menampilkan elemen pada puncak stack

def isempty(): ==> Digunakan untuk mendefinisikan sebuah fungsi "isempty"

def tampilkan(stack): ==> Digunakan untuk mendefinisikan sebuah fungsi

"tampilkan"

Contoh stack pada python

```
1 tumpukan = [1,2,3,4,5,6,7,8]
2 print(tumpukan)
3
4 #memasukan data baru
5 tumpukan.append(9)
6 print("data masuk", 9)
7 print('data sekarang', tumpukan)
8
9 tumpukan.append(10)
10 print("data masuk", 10)
11 print('data sekarang', tumpukan)
12
13 dataKeluar = tumpukan.pop() # mengeluarkan data yang paling terakhir yaitu 10
```

Ln: 9, Col: 20



Run



Share

Command Line Arguments



[1, 2, 3, 4, 5, 6, 7, 8]



data masuk 9



data sekarang [1, 2, 3, 4, 5, 6, 7, 8, 9]



data masuk 10



data sekarang [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]



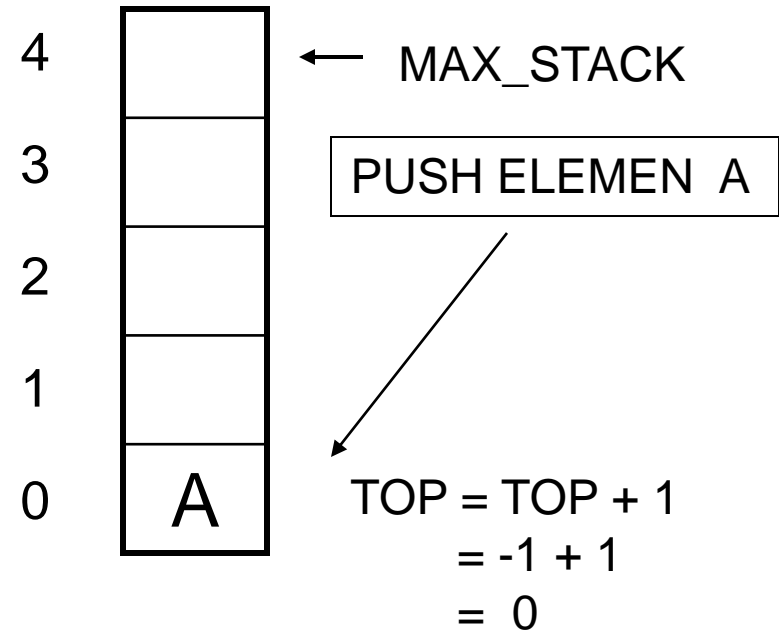
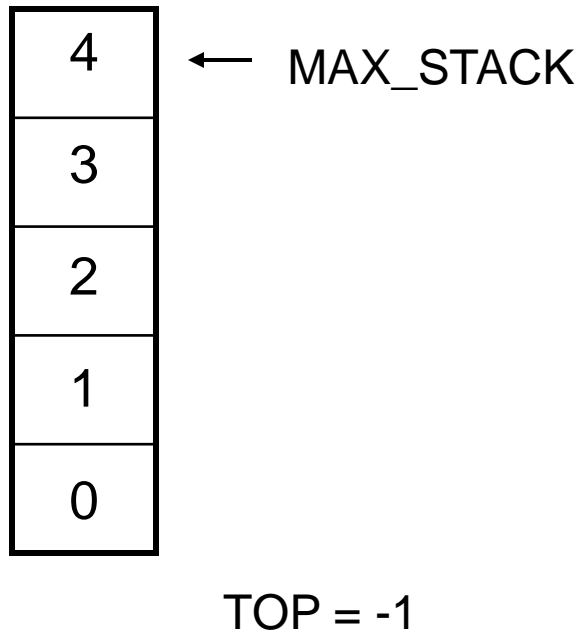
data yang keluar adalah : 10

data terakhir adalah : [1, 2, 3, 4, 5, 6, 7, 8, 9]

Fungsi PUSH

- Digunakan untuk memasukkan elemen ke dalam stack dan selalu menjadi elemen teratas stack
- Dengan cara :
 1. Menambah satu (increment) nilai TOP of STACK setiap ada penambahan elemen stack selama stack masih belum penuh
 2. Isikan nilai baru ke stack berdasarkan indeks TOP of STACK setelah ditambah satu (diincrement)

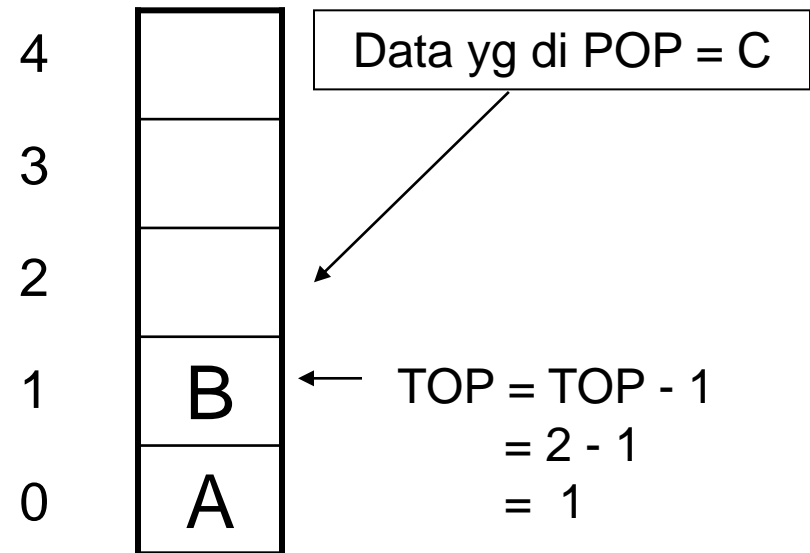
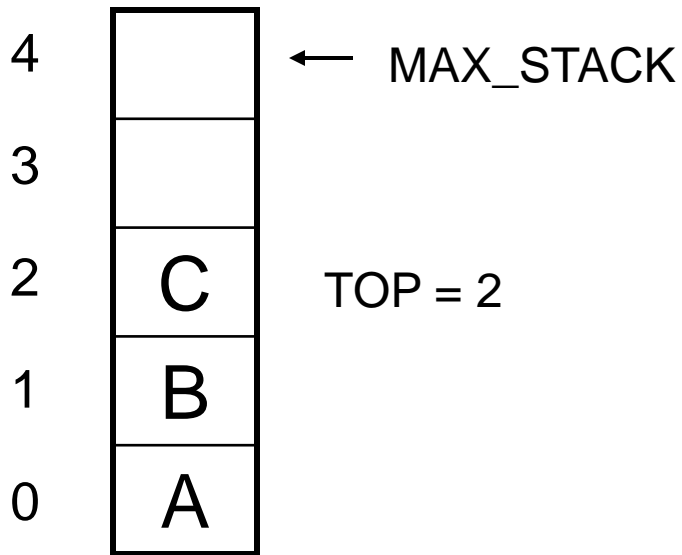
Fungsi PUSH (Lanjutan)



Fungsi POP

- Digunakan untuk menghapus elemen yang berada pada posisi paling atas dari stack.
- Dengan cara :
 1. Ambil dahulu nilai elemen teratas stack dengan mengakses TOP of STACK.
 2. Tampilkan nilai yang akan diambil.
 3. Lakukan decrement nilai TOP of STACK sehingga jumlah elemen stack berkurang 1

Fungsi POP (Lanjutan)



Latihan

Diketahui suatu stack dgn $\text{max_stack} = 6$

1. Bila dilakukan PUSH 3 elemen kedalam stack, kemudian di PUSH lagi 2 elemen dan di POP 3 elemen. Maka dimana posisi Top of Stack ?
2. IsEmpty pada kondisi terakhir adalah ?
3. Dari kondisi diatas, Berapa elemen yg hrs di PUSH unt mencapai kondisi penuh Top of Stack = max_stack ?
4. Berapa elemen yg hrs di POP unt mencapai kondisi Empty = True

Pertemuan 9

QUEUE (ANTREAN)



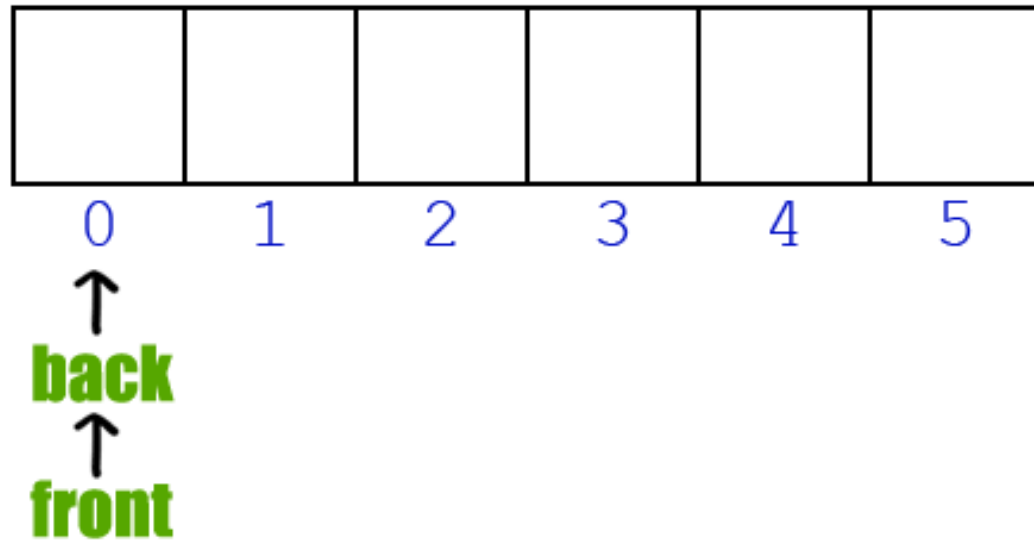
PENGERTIAN QUEUE (ANTREAN)

Struktur Data Antrean (Queue) adalah suatu bentuk khusus dari List Linier dengan operasi pemasukan data hanya diperbolehkan pada salah satu sisi, yang disebut sisi Belakang / ekor (Tail) dan operasi penghapusan hanya diperbolehkan pada sisi lainnya yang disebut sisi Depan / kepala (Head) dari LinkedList.

Prinsip Antrean : FIFO (First In First Out)
FCFS (First Come First Serve)

“Yang Tiba lebih awal Maka akan dilayani Terlebih Dahulu”

Deklarasi Queue



Queue dengan Python

```
1 from collections import deque # import module
2 antrian = deque([1,2,3,4,5])
3 print('data antrian sekarang : ', antrian)
4
5 #menambahkan data antrian
6 antrian.append(6)
7 print('data antrian masuk : 6')
8 print('data antrian : ', antrian)
9
10 #mengurangi antrian
11 keluar = antrian.popleft()
12 print('data keluar : ', keluar)
13 print('data antrian sekarang : ', antrian)
```

Hasil



data antrian sekarang : deque([1, 2, 3, 4, 5])



data antrian masuk : 6



data antrian : deque([1, 2, 3, 4, 5, 6])

data keluar : 1



data antrian sekarang : deque([2, 3, 4, 5, 6])

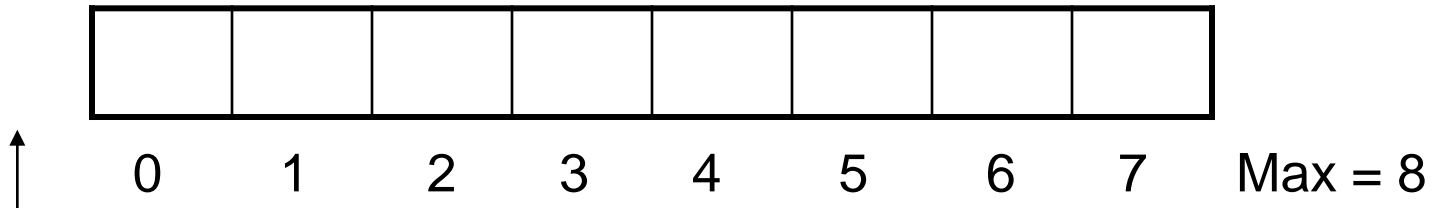
OPERASI QUEUE

- **PEEK ()**
Digunakan untuk mendapatkan elemen yang terdapat pada posisi terakhir tanpa menghapus element tersebut.
- **ISEMPTY()**
Untuk memeriksa apakah queue kosong
- **SIZE()**
mengembalikan jumlah item di dalam list. Tidak memerlukan parameter dan mengembalikan suatu integer.
- **ENQUEUE()**
Untuk menambahkan item pada posisi paling belakang
- **DEQUEUE()**
Untuk menghapus item dari posisi paling depan

Inisialisasi Head

Digunakan untuk membentuk dan menunjukan awal terbentuknya suatu Antrean / Queue

```
def __init__(self):  
    self.head = Node("head")  
    self.size = 0
```



Antrian pertama kali

Fungsi IsEmpty

- Untuk memeriksa apakah Antrian penuh atau kosong
- Head adalah tanda untuk kepala antrian (elemen pertama dalam antrian) yang tidak akan berubah-ubah
- Pergerakan pada Antrian terjadi dengan penambahan elemen Antrian kebelakang,

Fungsi isEmpty (Lanjutan)

```
def isEmpty(self):  
    return self.size == 0
```



0

1

2

3

4

5

6

7

Max = 8

Antrian kosong

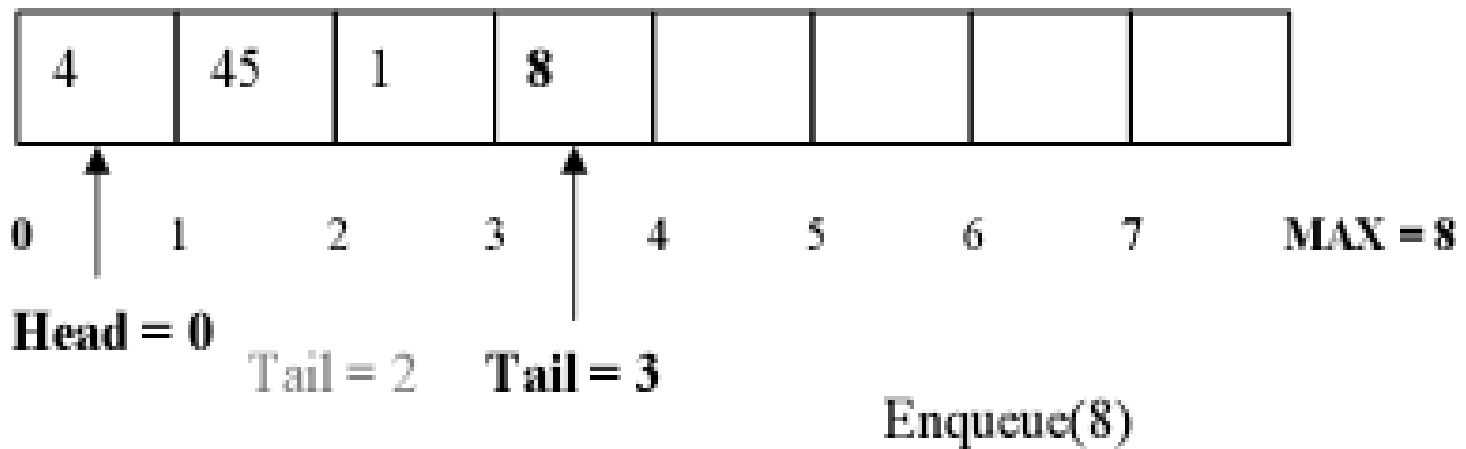
Fungsi Peek

Periksa antrian untuk melihat apakah ada antrian kosong

```
def peek(self):  
  
    if self.isEmpty():  
        raise Exception("Peeking from an empty stack")  
    return self.head.next.value
```

Fungsi Enqueue

- Untuk menambahkan elemen ke dalam Antrian, penambahan elemen selalu dilakukan pada elemen paling **belakang**
- Penambahan elemen selalu menggerakkan variabel Tail dengan cara menambahkan Tail terlebih dahulu



Fungsi Enqueue (Lanjutan)

Menambahkan nilai pada antrian

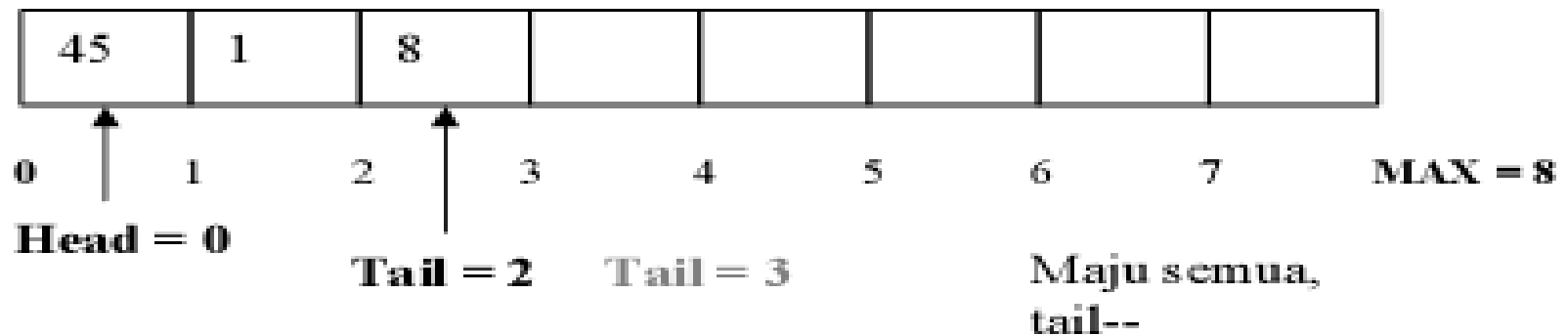
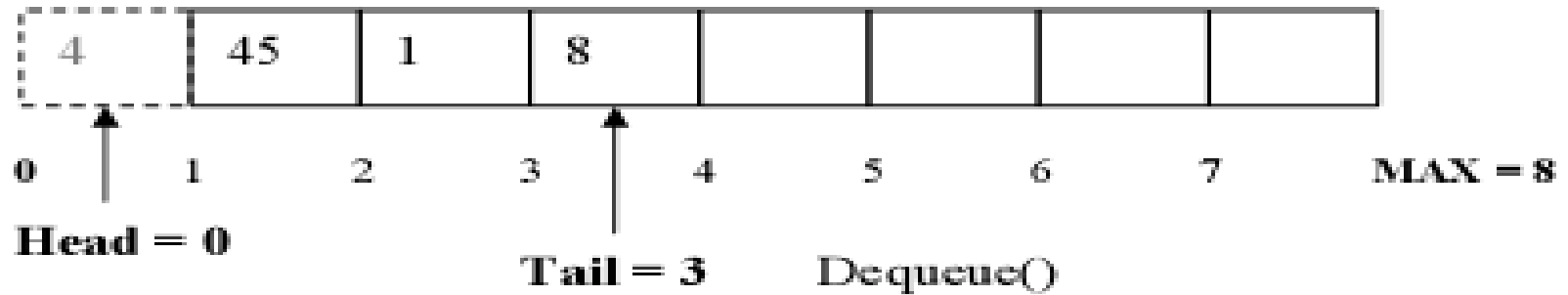
```
def enqueue(self, value):  
    node = Node(value)  
    node.next = self.head.next  
    self.head.next = node  
    self.size += 1
```

Fungsi Dequeue

- Digunakan untuk menghapus elemen terdepan (head) dari Antrian
- Dengan cara : menggeser semua elemen antrian kedepan dan mengurangi Tail dgn 1. Penggeseran dilakukan dengan menggunakan looping

```
def dequeue(self):  
    if self.isEmpty():  
        raise Exception("dequeue from an empty stack")  
    remove = self.head.next  
    self.head.next = self.head.next.next  
    self.size -= 1  
    return remove.value
```

Fungsi Dequeue (Lanjutan)



Fungsi SIZE

digunakan untuk mengetahui banyak elemen atau data yang ada di dalam list.

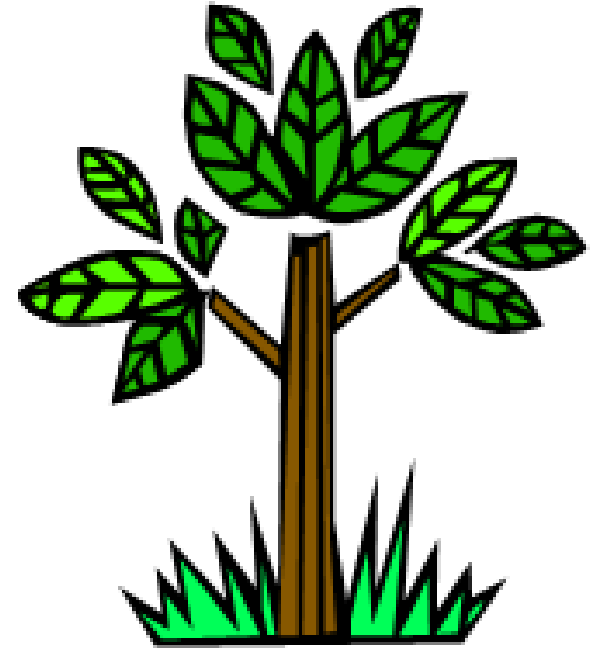
```
def getSize(self):  
    return self.size
```

Latihan

Berikan gambaran/ilustrasi dari kasus antrian berikut :

- a) Diketahui suatu Antrian/queue dgn $\text{max} = 6$.
- b) Lakukan Enqueue 4 elemen ke dalam antrian, dimanakah posisi Head dan Tail ?
- c) Kemudian lakukan Dequeue 2 elemen dari antrian. Maka dimana posisi Head dan Tail ?
- d) Dari keadaan diatas, bagaimanakah kondisi penuh dan IsEmpty nya ?

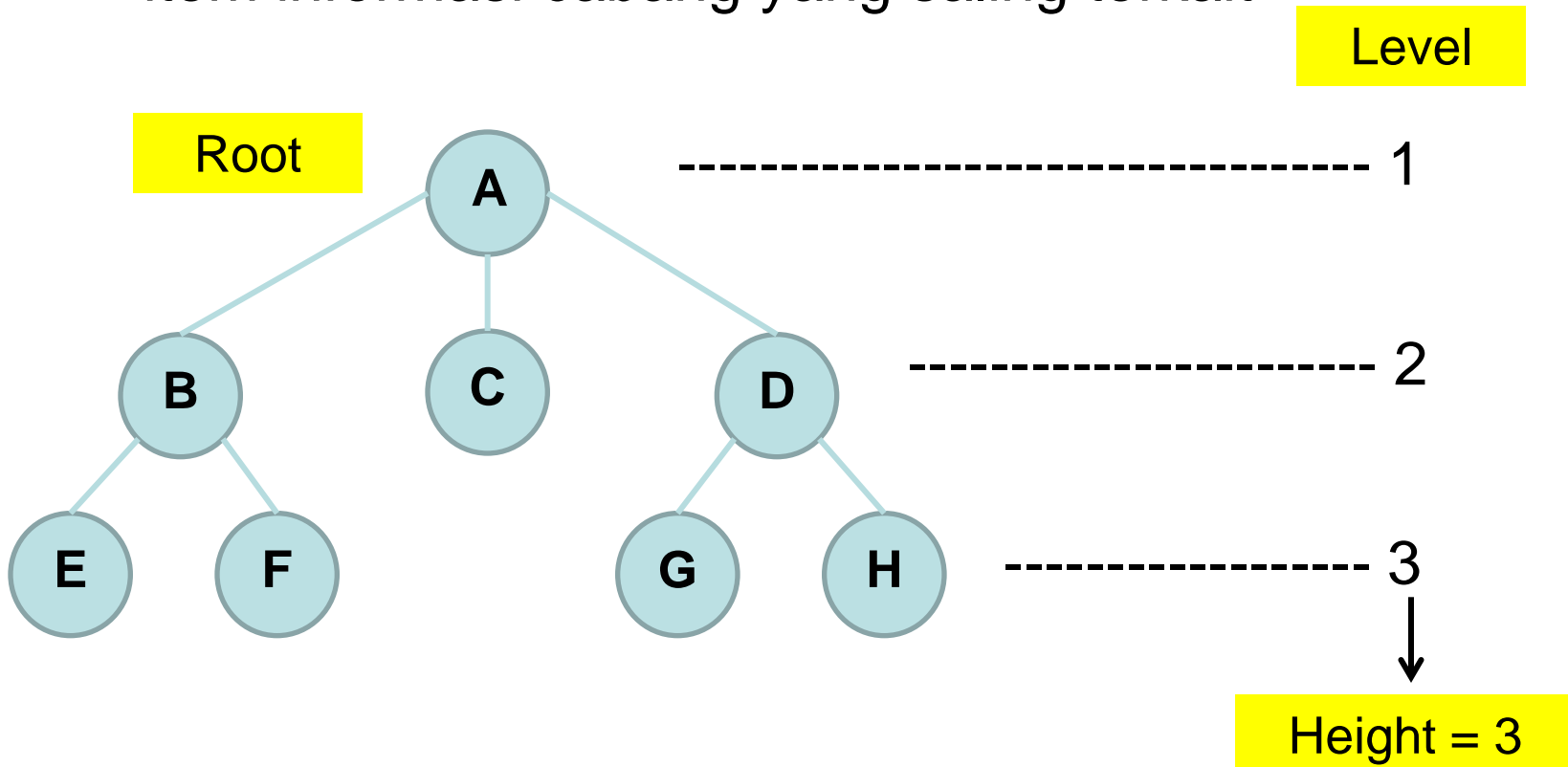
Pertemuan 10



STRUKTUR POHON & KUNJUNGAN POHON BINER

DEFINISI POHON (TREE)

- Pohon (Tree) termasuk struktur non linear yang didefinisikan sebagai data yang terorganisir dari suatu item informasi cabang yang saling terkait



Istilah – istilah Dalam Pohon

1. Predesesor

Node yang berada diatas node tertentu.
(contoh : B predesesor dari E dan F)

2. Succesor

Node yang berada dibawah node tertentu.
(contoh : E dan F merupakan succesor dari B)

3. Ancestor

Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama.
(contoh : A dan B merupakan ancestor dari F)

4. **Descendant**

Seluruh node yang terletak sesudah node tertentu dan terletak pada jalur yang sama.

(contoh : F dan B merupakan ancestor dari A)

5. **Parent**

Predesor satu level diatas satu node

(contoh : B merupakan parent dari F)

6. **Child**

Succesor satu level dibawah satu node

(contoh : F merupakan child dari B)

7. **Sibling**

Node yang memiliki parent yang sama dengan satu node (contoh : E dan F adalah sibling)

8. Subtree

Bagian dari tree yang berupa suatu node beserta descendant-nya (contoh : Subtree B, E, F dan Subtree D, G, H)

9. Size

Banyaknya node dalam suatu tree (contoh : gambar tree diatas memiliki size = 8)

10. Height

Banyaknya tingkat/level dalam suatu tree (contoh : gambar tree diatas memiliki height = 3)

11. Root (Akar)

Node khusus dalam tree yang tidak memiliki predesesor (Contoh : A)

12. Leaf (Daun)

Node-node dalam tree yang tidak memiliki daun (contoh : Node E,F,C,G,H)

13. Degree (Derajat)

Banyaknya child yang dimiliki oleh suatu node (contoh : Node A memiliki derajat 3, node B memiliki derajat 2)

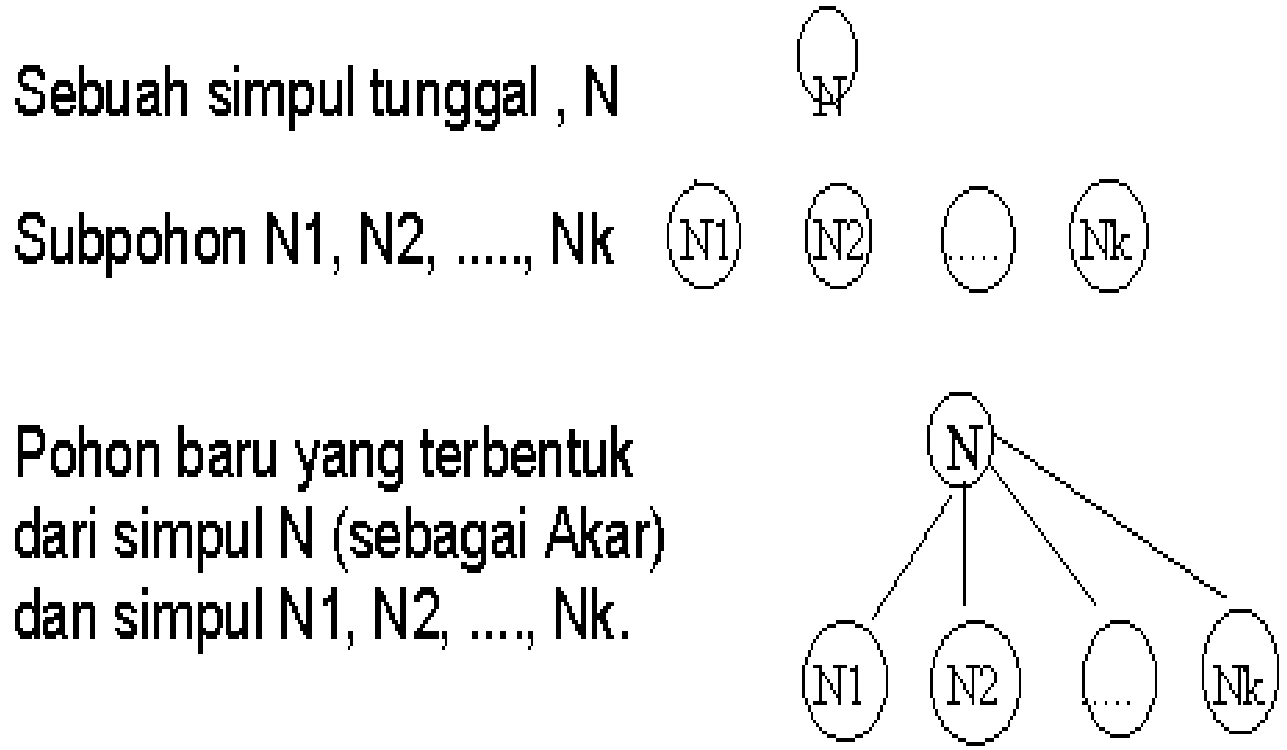
ISTILAH-ISTILAH DASAR

Pohon atau Tree adalah salah satu bentuk Graph terhubung yang tidak mengandung sirkuit.

Karena merupakan Graph terhubung, maka pada Pohon (Tree) selalu terdapat Path atau Jalur yang menghubungkan setiap simpul dalam dua pohon.

Pohon (Tree) dapat juga didefinisikan sebagai kumpulan elemen yang salah satu elemennya disebut dengan Akar (Root) dan sisa elemen lain (Simpul) yang terpecah menjadi sejumlah himpunan yang saling tidak berhubungan yang disebut dengan Subpohon (Subtree) atau cabang

Gambar berikut ini menjelaskan tentang pembentukan awal dari Pohon (Tree).



Sifat Utama Pohon Berakar

1. Jika Pohon mempunyai Simpul sebanyak n , maka banyaknya ruas atau edge adalah $(n-1)$.
2. Mempunyai Simpul Khusus yang disebut Root, jika Simpul tersebut memiliki derajat keluar ≥ 0 , dan derajat masuk = 0.
3. Mempunyai Simpul yang disebut sebagai Daun / Leaf, jika Simpul tersebut berderajat keluar = 0, dan berderajat masuk = 1.
4. Setiap Simpul mempunyai Tingkatan / Level yang dimulai dari Root yang Levelnya = 1 sampai dengan Level ke - n pada daun paling bawah. Simpul yang mempunyai Level sama disebut Bersaudara atau Brother atau Stribling.

5. Pohon mempunyai Ketinggian atau Kedalaman atau Height, yang merupakan Level tertinggi
6. Pohon mempunyai Weight atau Berat atau Bobot, yang banyaknya daun (leaf) pada Pohon.
7. Banyaknya Simpul Maksimum sampai Level N adalah :

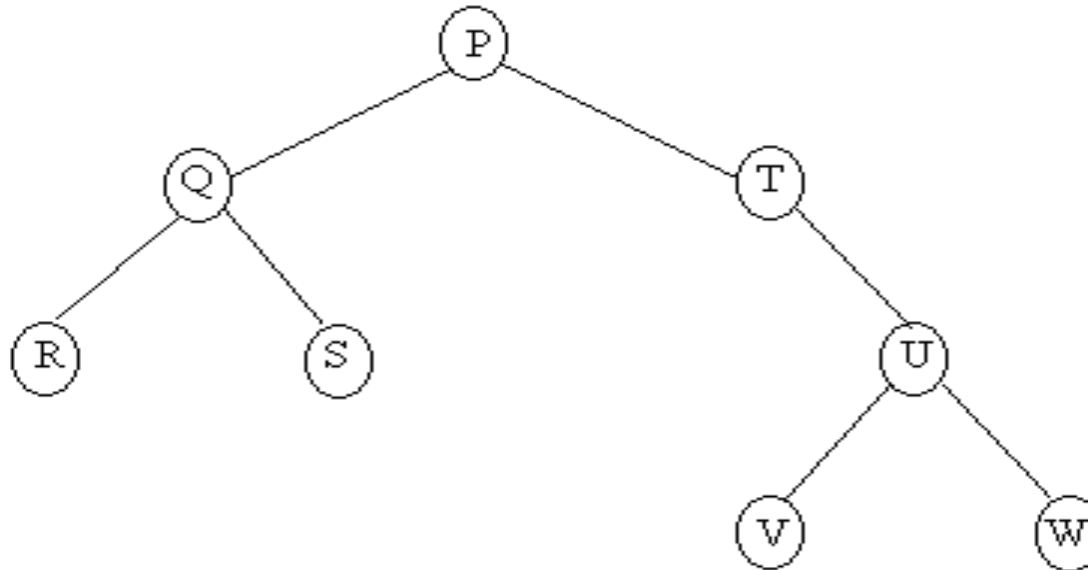
$$2^N - 1$$

8. Banyaknya Simpul untuk setiap Level I adalah :

$$\sum_{I=1}^N 2^{(I-1)}$$

Hutan (Forest) adalah kumpulan Pohon yang tidak saling berhubungan

Diketahui suatu bentuk Pohon Berakar T sebagai berikut :



Pohon Diatas Mempunyai :

- a. Simpul sebanyak = 8 dan edge = $n - 1 = 8 - 1 = 7$
- b. Root pada Pohon T diatas adalah Simpul P
- c. Mempunyai daun (Leaf) = 4, yaitu = R, S, V dan W

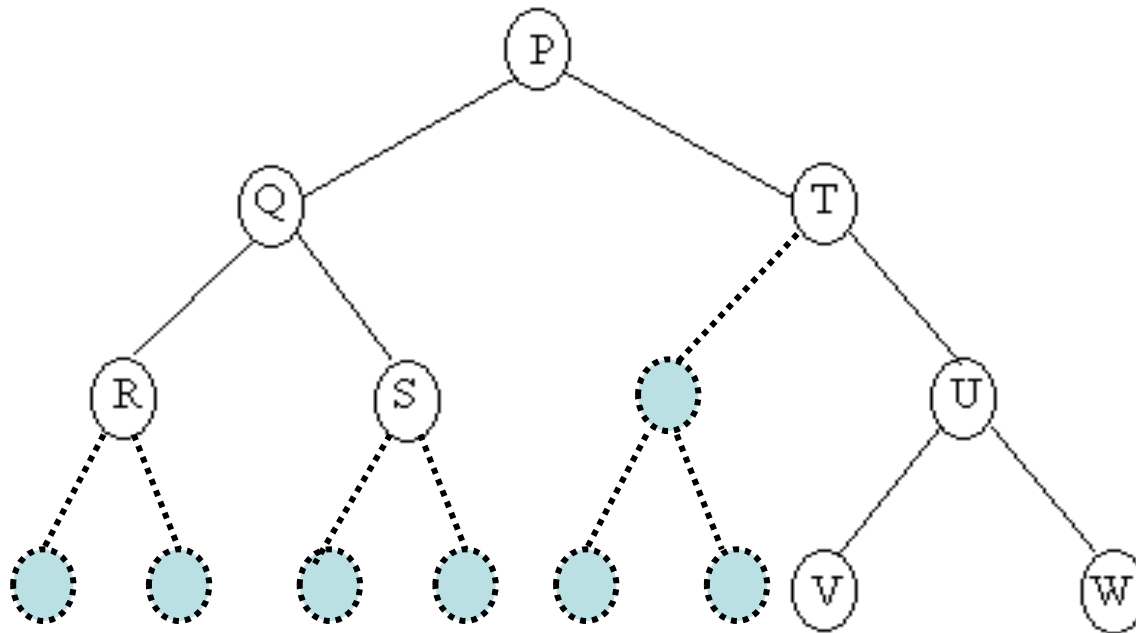
- d. Level (tingkatan) Pohon = 4 yaitu :
 - Level 1 = Simpul P
 - Level 2 = Simpul Q dan T
 - Level 3 = Simpul R, S dan U
 - Level 4 = Simpul V dan W
- e. Ketinggian atau kedalaman = jumlah level = 4
- f. Weight atau berat atau bobot = jumlah daun = 4

Dalam gambar Pohon T diatas dapat dibentuk 2 buah hutan (forest), bila simpul P dihilangkan, yaitu :

Hutan 1 : Q,R,S

Hutan 2 : T,U,V,W

- g. Banyaknya Simpul Maksimum yang dapat terbentuk sampai Level 4 (bila simpul pada pohon dianggap penuh) adalah : $2^N - 1$
 $2^{(4)} - 1 = 16 - 1 = 15$



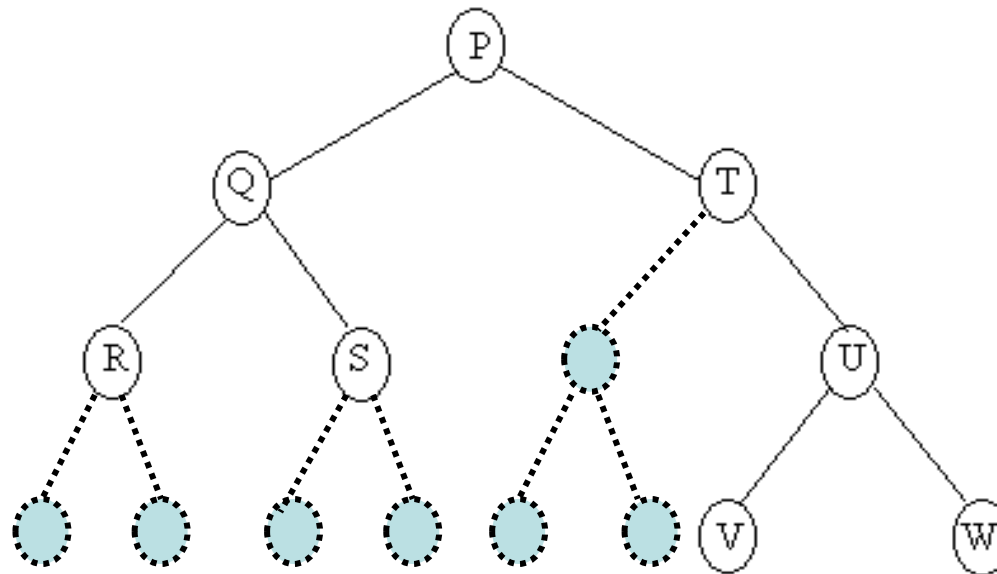
h. Banyaknya Simpul maksimum untuk setiap Level I (bila simpul pada pohon dianggap penuh) adalah :

Maksimum Simpul pada level 2 = $2^{(1-1)}$

$$= 2^{(2-1)} = 2$$

Maksimum Simpul pada level 3 = $2^{(3-1)} = 4$

Maksir



POHON BINAR (BINARY TREE)

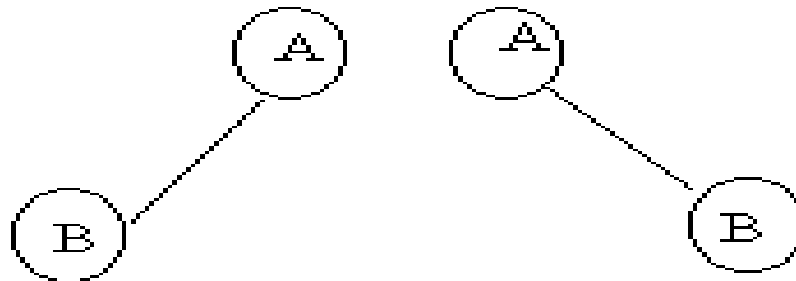
Struktur ini biasanya digunakan untuk menyajikan data yang mengandung hubungan hirarkial antara elemen-elemennya.

Bentuk Pohon Berakar yang lebih mudah dikelola dalam komputer adalah Pohon Biner (Binary Tree) yang lebih dikenal sebagai Pohon Umum (General Tree) yang dapat didefinisikan sebagai kumpulan simpul yang mungkin kosong atau mempunyai akar dan dua Subpohon yang saling terpisah yang disebut dengan Subpohon Kiri / cabang kiri (Left Subtree) dan Subpohon Kanan / cabang kanan (Right Subtree).

Karakteristik Pohon Binar (Binary Tree) :

1. Setiap Simpul paling banyak hanya memiliki dua buah anak
2. Derajat Tertinggi dari setiap Simpul adalah dua.
3. Dibedakan antara Cabang Kiri dan Cabang Kanan.
4. Dimungkinkan tidak mempunyai Simpul

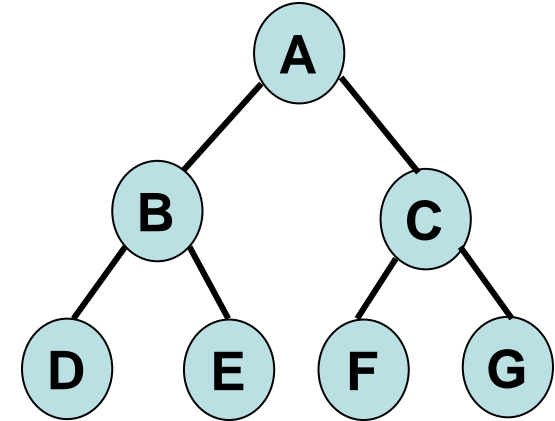
Berikut ini diberikan contoh gambar Pohon Binar (Binary Tree) dengan *Cabang Kiri* dan *Cabang Kanan*.



ISTILAH PADA POHON BINER

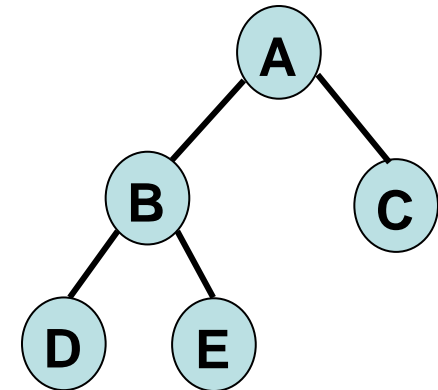
- **Pohon Biner Penuh (Full Binary Tree)**

Semua simpul (kecuali daun) memiliki 2 anak dan tiap cabang memiliki panjang ruas yang sama



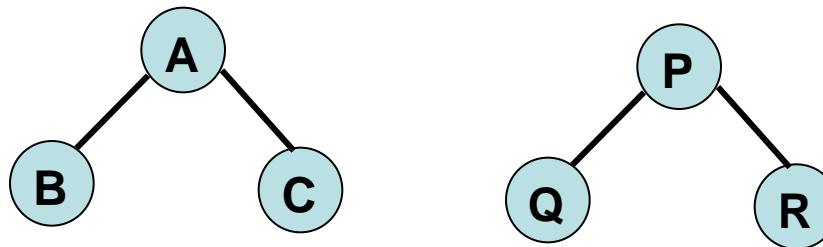
- **Pohon Biner Lengkap (Complete Binary Tree)**

Hampir sama dengan Pohon Biner Penuh, semua simpul (kecuali daun) memiliki 2 anak tetapi tiap cabang memiliki panjang ruas berbeda



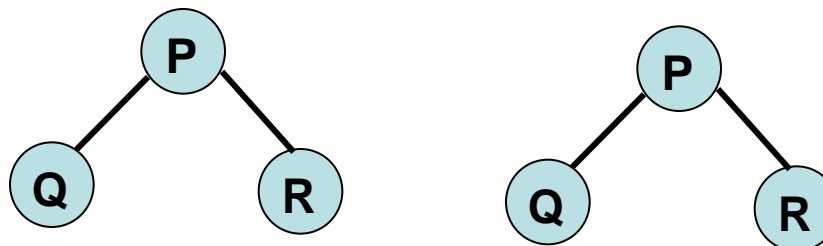
- **Pohon Biner Similer**

Dua pohon yang memiliki struktur yang sama tetapi informasinya berbeda



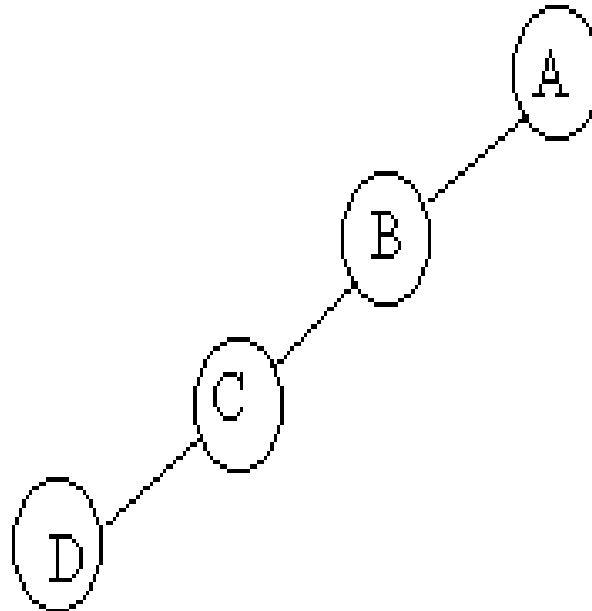
- **Pohon Biner Ekuivalent**

Dua pohon yang memiliki struktur dan informasi yang sama



- **Pohon Biner Miring (Skewed Tree)**

Dua pohon yang semua simpulnya mempunyai satu anak / turunan kecuali daun



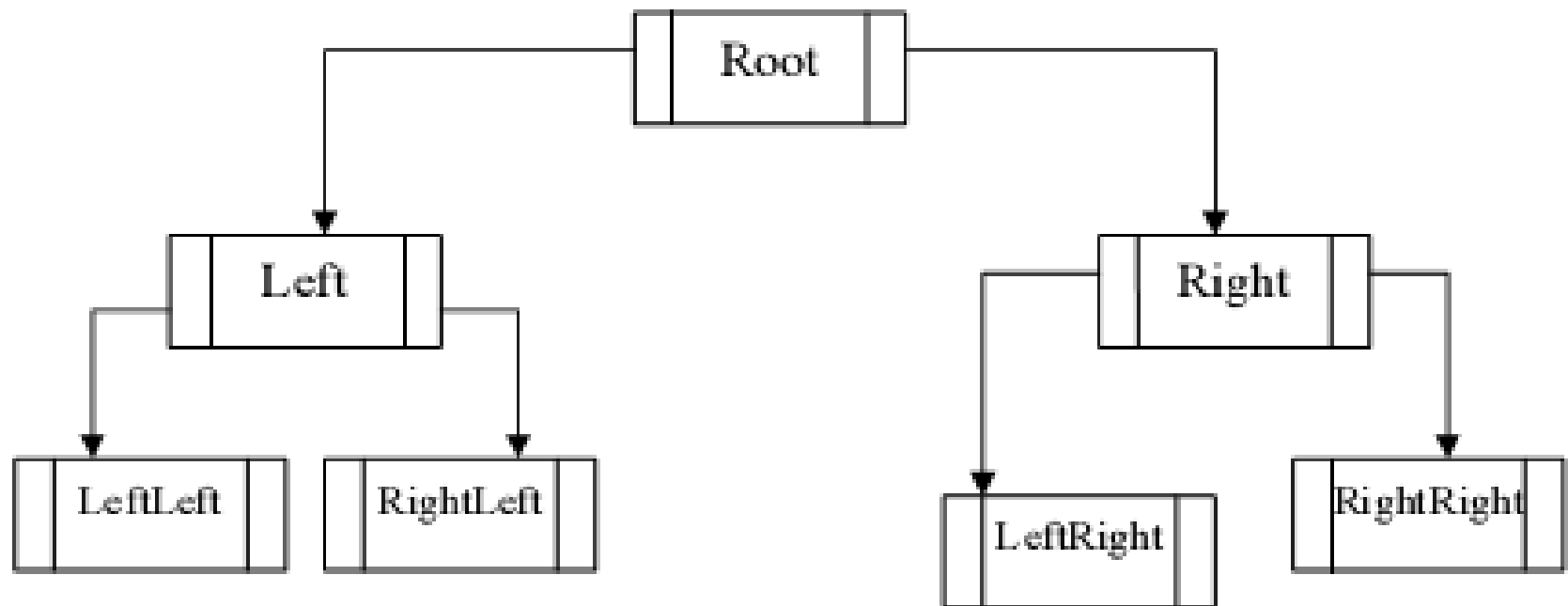
Deklarasi Pohon Biner

Dalam setiap simpul selalu berisi dua buah Pointer untuk menunjuk ke cabang Kiri dan cabang Kanan dan informasi yang akan disimpan dalam simpul tersebut.

Definisi Node

```
# The Node Class defines the structure of a Node
class Node:
# Initialize the attributes of Node
def __init__(self, data):
self.left = None # Left Child
self.right = None # Right Child
self.data = data # Node Data
```

Ilustrasi:



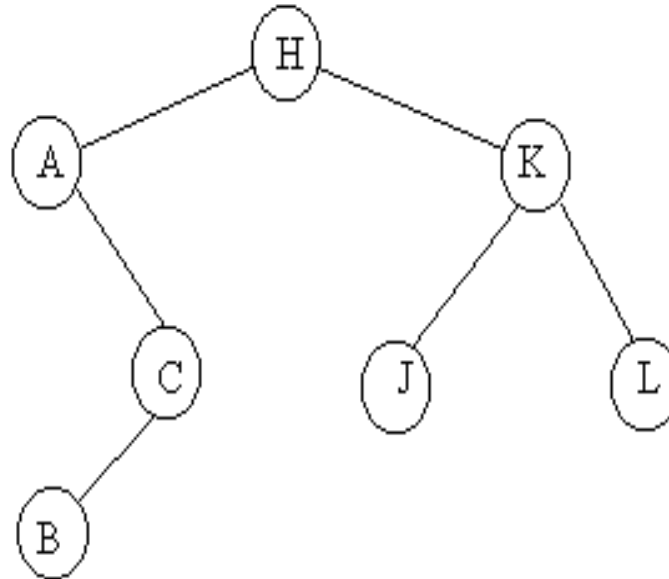
Penyajian Pohon Binar (Binary Tree)

- Tree dapat dibuat dengan menggunakan linked list secara rekursif.
- Linked list yang digunakan adalah double linked list non circular
- Data yang pertama kali masuk akan menjadi node root.
- Data yang lebih kecil dari data node root akan masuk dan menempati node kiri dari node root, sedangkan jika lebih besar dari data node root, akan masuk dan menempati node di sebelah kanan node root.

Bila diberikan untai HAKJCBL, maka proses untuk dapat membentuk pohon biner dari untai diatas adalah :

1. Karakter pertama 'H' ditempatkan sebagai akar (root)
2. Karakter 'A', karena lebih kecil dari 'H', maka akan menempati cabang kiri.
3. Karakter 'K', karena lebih besar dari 'H', maka akan menempati cabang kanan.
4. Karakter 'J', lebih besar dari 'H' dan kecil dari 'K', maka menempati cabang kiri 'K'.
5. Karakter 'C', karena lebih besar dari 'A', maka akan menempati cabang kanan.
6. Karakter 'B', karena lebih kecil dari 'C', maka akan menempati cabang kiri.
7. Karakter 'L', lebih besar dari 'K', maka menempati cabang kiri kanan.

Sehingga terbentuk pohon biner seperti berikut :



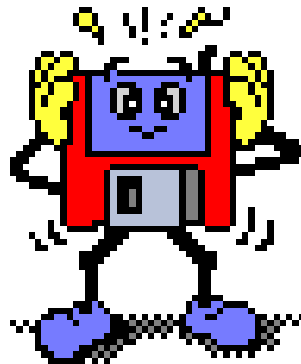
Latihan

Buatlah pohon biner dari barisan bilangan berikut:

1. 12, 22, 8, 19, 10, 9, 20, 4, 2, 6
2. 2, 3, 4, 5, 50, 10, 15, 13, 20, 12, 10, 7
3. 7, 13, 4, 6, 5, 9, 15, 20, 60, 14, 40, 70
4. 50, 45, 55, 41, 49, 13, 60, 70, 40, 35, 30, 20, 80, 75, 85
5. 12, 19, 11, 17, 29, 21, 20, 22, 13, 14, 18, 16, 15

Pertemuan 11

KUNJUNGAN PADA POHON BINER



Kunjungan Pohon Biner

Kunjungan pada Pohon Binar merupakan salah satu operasi yang sering dilakukan pada suatu Pohon Binar tepat satu Kali (Binary Tree Traversal). Operasi ini terbagi menjadi 3 bentuk :

1. Kunjungan secara Preorder (Depth First Order), mempunyai urutan :
 - a. Cetak isi simpul yang dikunjungi (Simpul Akar)
 - b. Kunjungi Cabang Kiri
 - c. Kunjungi Cabang Kanan

Kunjungan Pohon Biner (Lanjutan)

2. Kunjungan secara Inorder (Symetric Order), mempunyai urutan :
 - a. Kunjungi Cabang Kiri
 - b. Cetak isi simpul yang dikunjungi (Simpul Akar)
 - c. Kunjungi Cabang Kanan

3. Kunjungan secara Postorder, mempunyai urutan :
 - a. Kunjungi Cabang Kiri
 - b. Kunjungi Cabang Kanan
 - c. Cetak isi simpul yang dikunjungi (Simpul Akar)

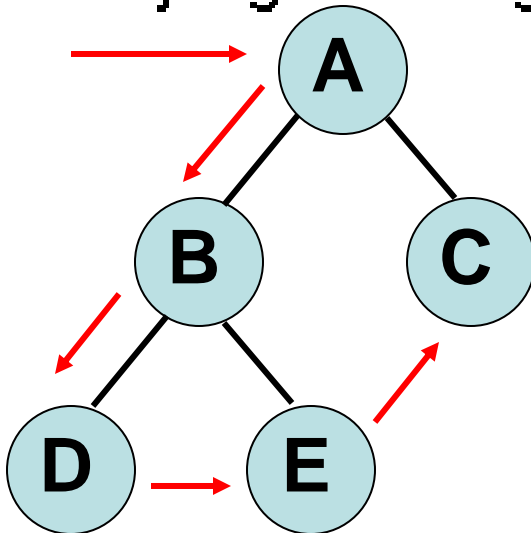
Kunjungan Pohon Biner (Lanjutan)

Pada ketiga cara kunjungan diatas, kunjungan ke Cabang Kiri dilakukan terlebih dahulu, baru kemudian kunjungan ke Cabang Kanan. Dengan orientasi semacam ini, Ketiga kunjungan diatas disebut dengan Left To Right Oriented (LRO).

Jika kunjungan ke Cabang Kanan dilakukan lebih dahulu baru kemudian kunjungan ke Cabang Kiri, maka Orientasi semacam ini disebut Right To Left Oriented (RLO).

PreOrder

1. Kunjungan secara Preorder (Depth First Order), mempunyai urutan :
 - a. Cetak isi simpul yang dikunjungi (Simpul Akar)
 - b. Kunjungi Cabang Kiri
 - c. Kunjungi Cabang Kanan

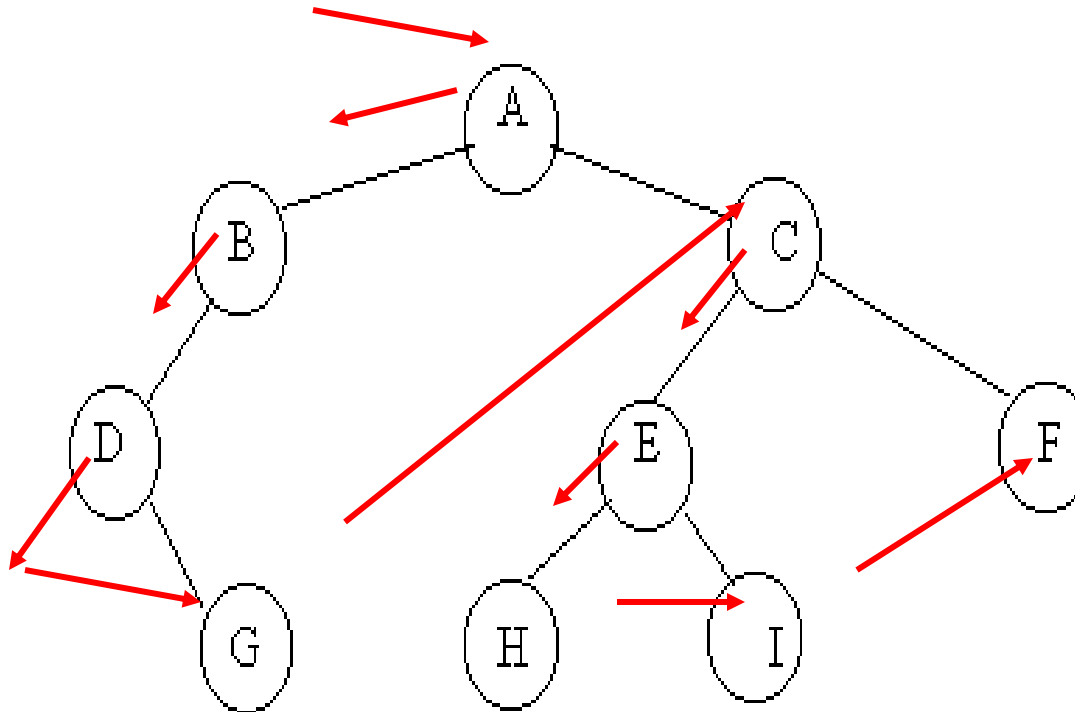


A B D E C

Klik Animasi

Contoh PreOrder

Kunjungan PreOrder



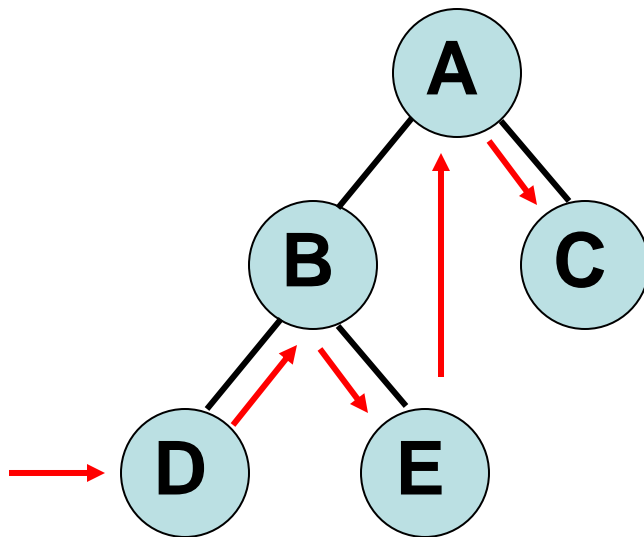
Hasil : A B D G C E H I F

Klik Animasi

InOrder

2. Kunjungan secara Inorder (Symetric Order), mempunyai urutan :

- Kunjungi Cabang Kiri
- Cetak isi simpul yang dikunjungi (Simpul Akar)
- Kunjungi Cabang Kanan

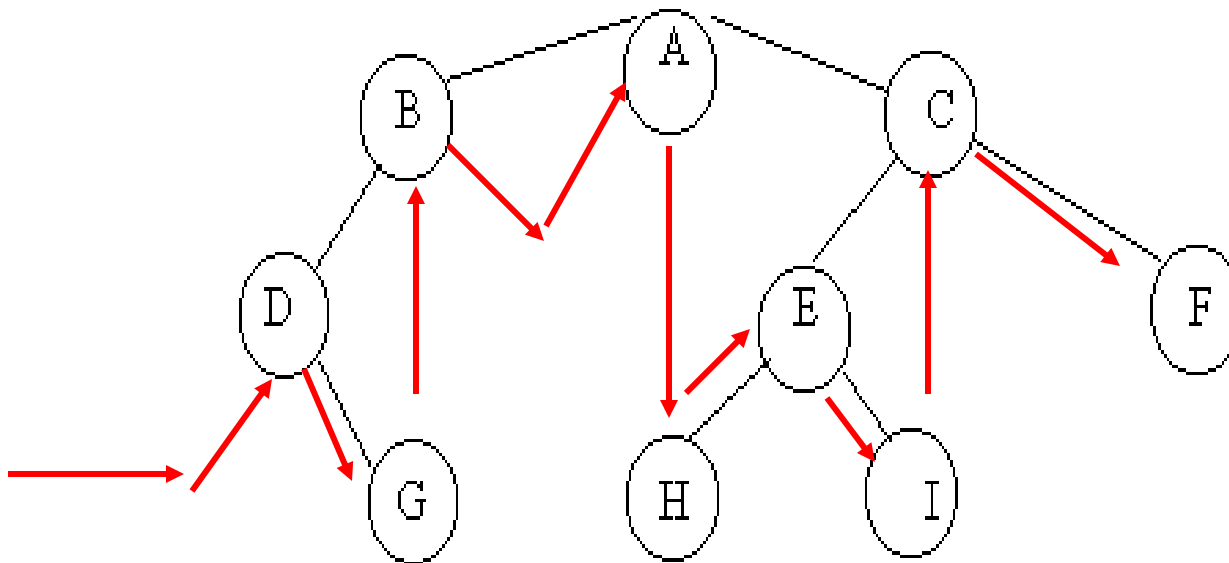


D B E A C

Klik Animasi

Contoh InOrder

Kunjungan InOrder

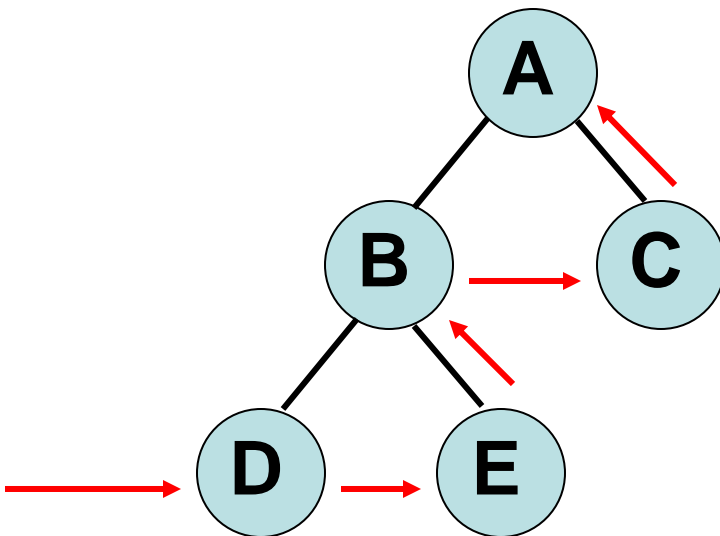


Hasil : D G B A H E I C F

Klik Animasi

PostOrder

3. Kunjungan secara Postorder, mempunyai urutan :
- Kunjungi Cabang Kiri
 - Kunjungi Cabang Kanan
 - Cetak isi simpul yang dikunjungi (Simpul Akar)

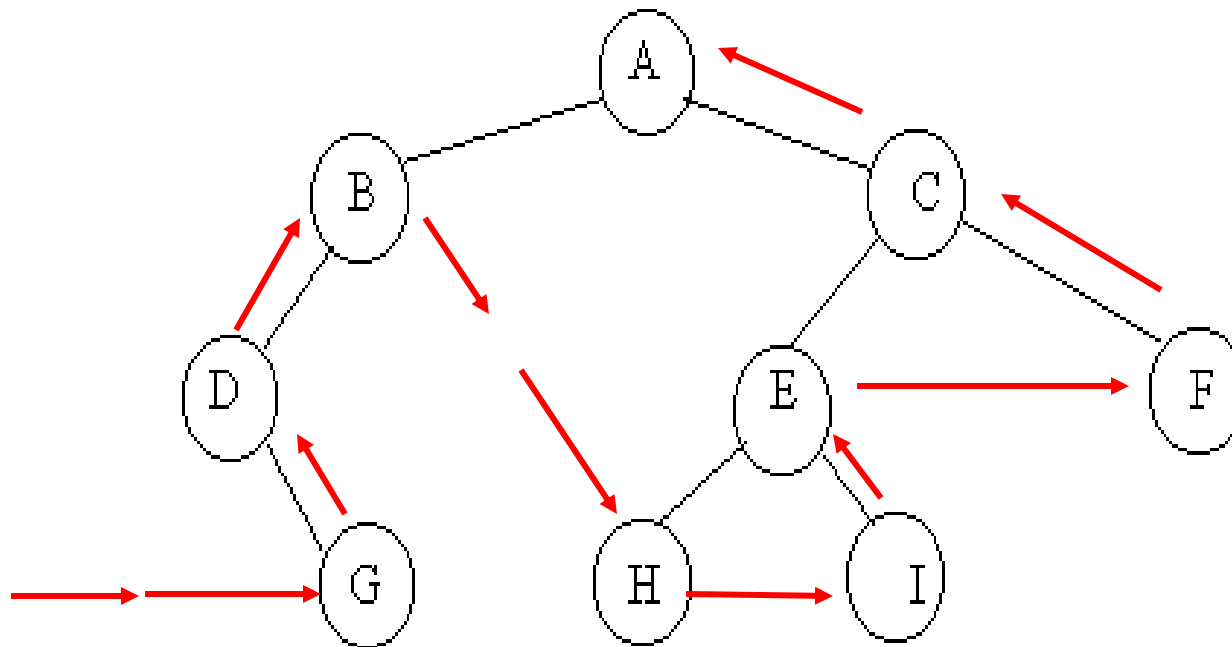


D E B C A

Klik Animasi

Contoh PostOrder

Kunjungan PostOrder



Hasil : G D B H I E F C A

Klik Animasi

Kunjungan LevelOrder

Selain kunjungan yang dijelaskan diatas, masih ada satu macam kunjungan masih ada satu macam kunjungan lagi yaitu kunjungan LevelOrder.

Kunjungan dimulai dari simpul yang ada pada tingkat 1 (Akar), diteruskan pada simpul di tingkat 2, tingkat 3 dan seterusnya.

Secara singkat kunjungan Level Order ini dapat dijelaskan sebagai berikut.

1. Dimulai dengan memasukkan Akar kedalam antrean.
2. Kemudian mengeluarkan Akar tersebut keluar dari antrean.

Pada saat Akar tersebut dikeluarkan dari antrean, cabang kiri dan cabang kanan secara berturut-turut dimasukkan dalam antrean.

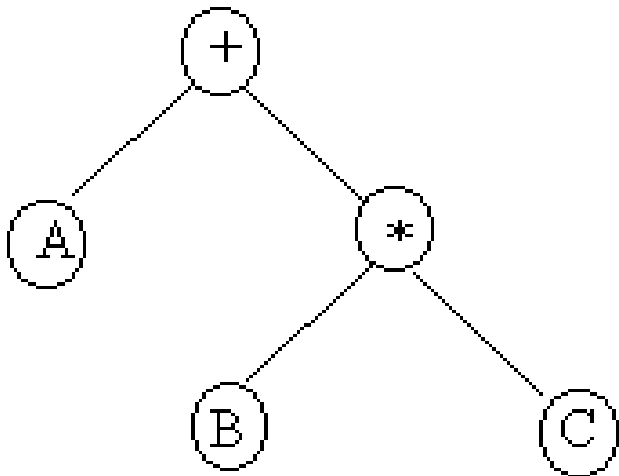
Dengan kata lain jika suatu elemen dikeluarkan dari antrean, maka cabang kiri dan kanan dari elemen yang baru saja dikeluarkan dimasukkan kedalam antrean.

APLIKASI POHON BINER

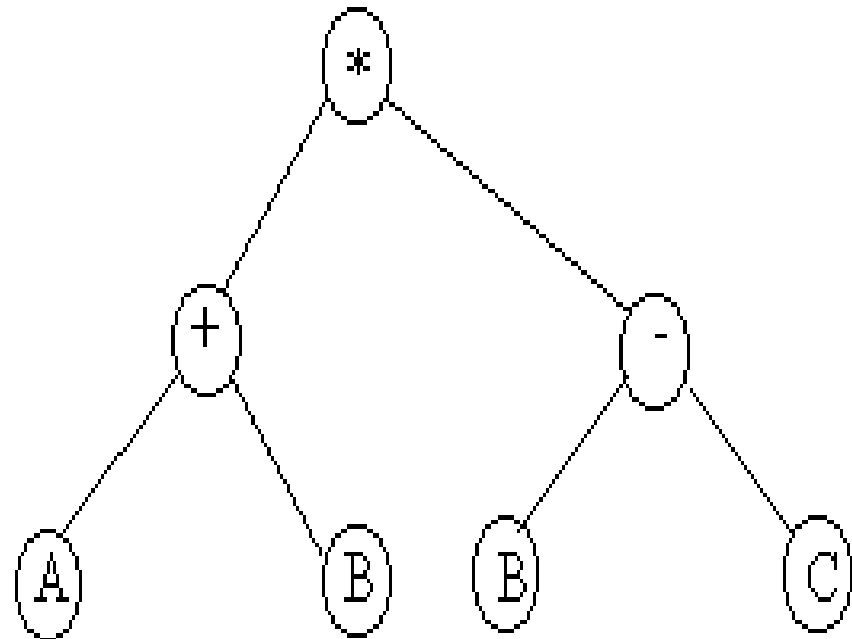
NOTASI PREFIX, INFIX DAN POSTFIX

Pada bagian ini akan dibahas tentang bagaimana menyusun sebuah Pohon Binar yang apabila dikunjungi secara PreOrder akan menghasilkan Notasi Prefix, kunjungan secara InOrder menghasilkan Notasi Infix, dan kunjungan PostOrder menghasilkan Notasi Postfix.

Aplikasi Pohon Biner (Lanjutan)

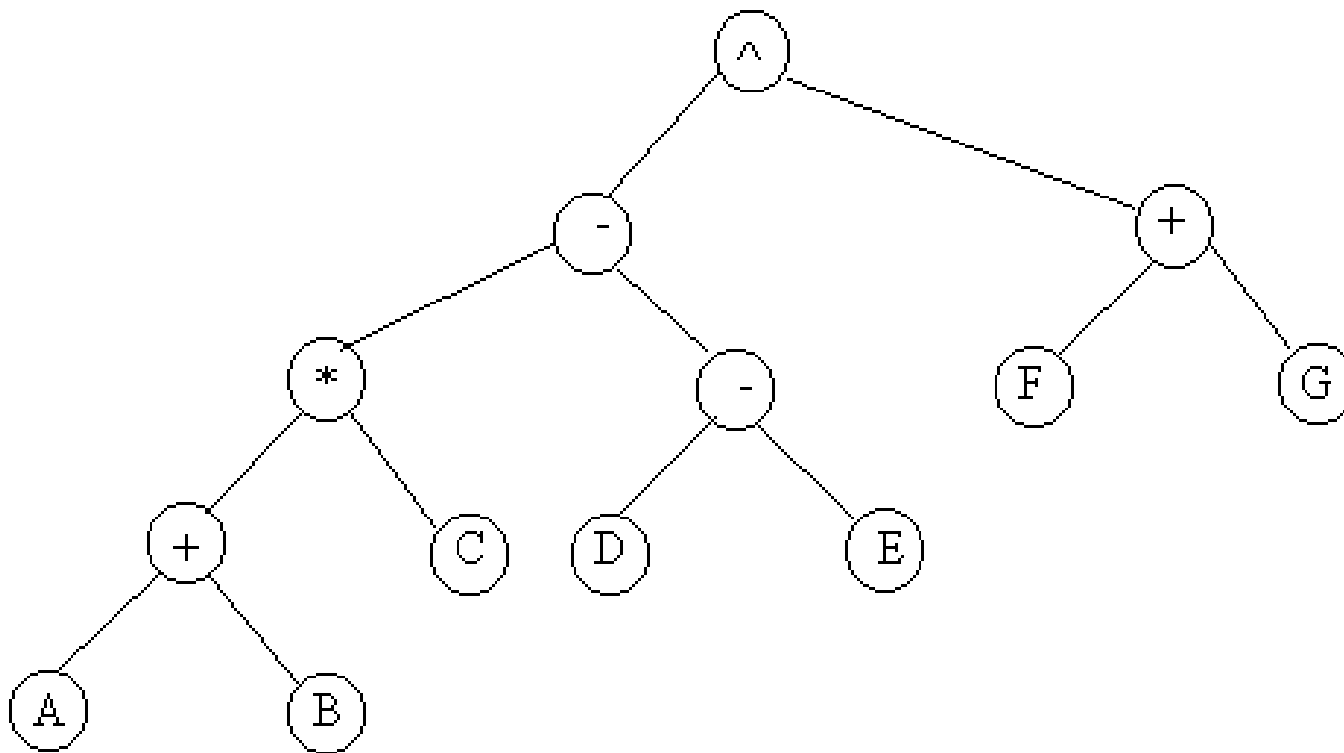


(a)



(b)

Aplikasi Pohon Biner (Lanjutan)



(c)

Aplikasi Pohon Biner (Lanjutan)

Berdasarkan Gambar diatas, apabila dilakukan kunjungan secara PreOrder, maka akan diperoleh Notasi Prefix dari persamaan-persamaan yang digambarkan tersebut, yaitu :

$+A*BC$ (Gambar.a)

$*+AB-BC$ (Gambar.b)

$\wedge-*+ABC-DE+FG$ (Gambar.c)

Jika dilakukan kunjungan secara InOrder, akan diperoleh Notasi Infixnya, yaitu :

$(A+(B*C))$ (Gambar.a)

$((A+B) * (B-C))$ (Gambar.b)

$(((((A+B) * C) - (D-E))^{\wedge}(F+G)))$ (Gambar.c)

Aplikasi Pohon Biner (Lanjutan)

Jika dilakukan kunjungan secara PostOrder, akan diperoleh Notasi Postfixnya, yaitu :

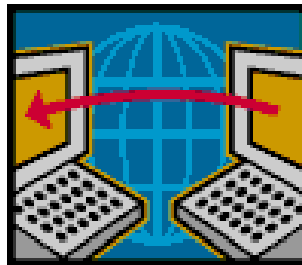
ABC^*+ (Gambar.a)

$AB+BC-^*$ (Gambar.b)

$AB+C^*DE--FG+^{\wedge}$ (Gambar.c)

Pertemuan 12

GRAPH, MATRIK PENYAJIAN GRAPH



GRAPH

Suatu Graph mengandung 2 himpunan, yaitu :

1. Himpunan V yang elemennya disebut simpul (Vertex atau Point atau Node atau Titik)
2. Himpunan E yang merupakan pasangan tak urut dari simpul. Anggotanya disebut Ruas (Edge atau rusuk atau sisi)

Graph seperti dimaksud diatas, ditulis sebagai $G(E,V)$.

Banyaknya simpul (vertex) disebut **Order**, sedangkan banyaknya ruas (edge) disebut **Size** dari Graph.

Contoh :

Gambar berikut menanyakan Graph $G(E,V)$ dengan :

1. V mengandung 4 simpul, yaitu simpul A, B, C, D .
2. E mengandung 5 ruas, yaitu :

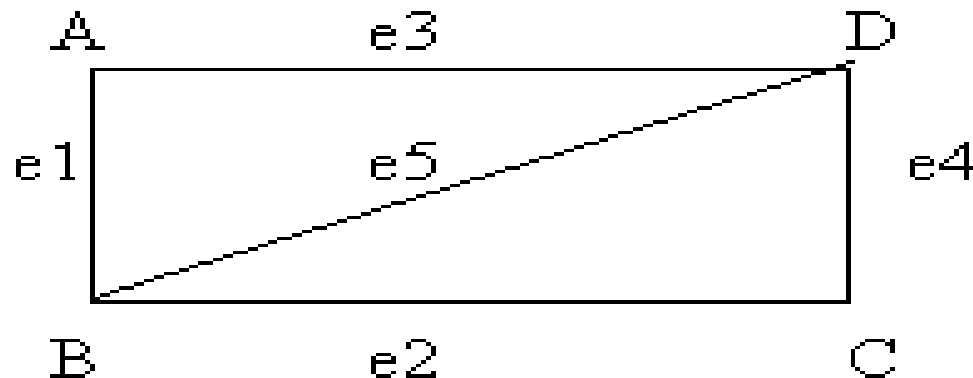
$$e1 = (A,B)$$

$$e4 = (C,D)$$

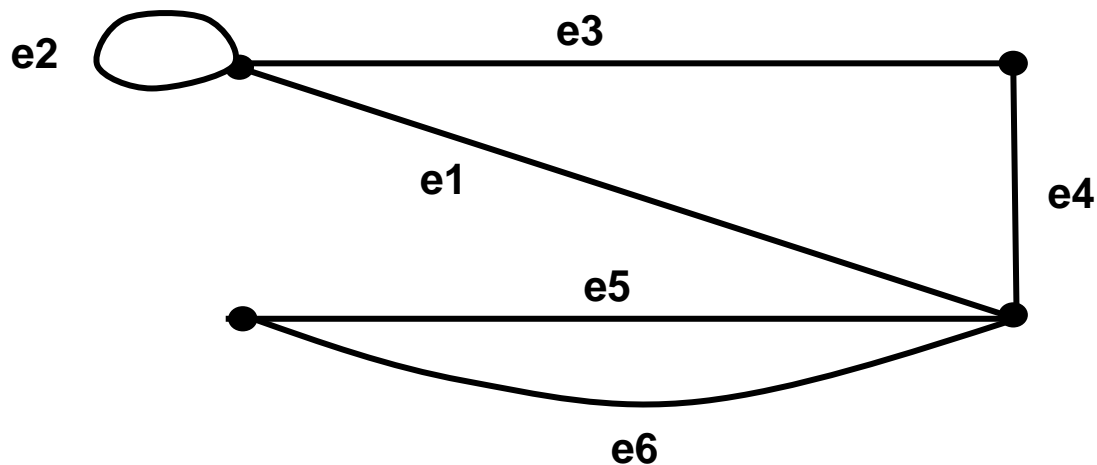
$$e2 = (B,C)$$

$$e5 = (B,D)$$

$$e3 = (A,D)$$



Gambar dibawah ini menyatakan suatu **Multigraph**. Disini, ruas e2 pada kedua titik ujungnya adalah simpul yang sama, yaitu simpul A. Ruas semacam ini disebut **Gelung atau Self-Loop**. Sedangkan ruas e5 dan e6 mempunyai titik ujung yang sama, yaitu simpul-simpul B dan C. Kedua ruas ini disebut ruas **berganda** atau ruas **sejajar**.

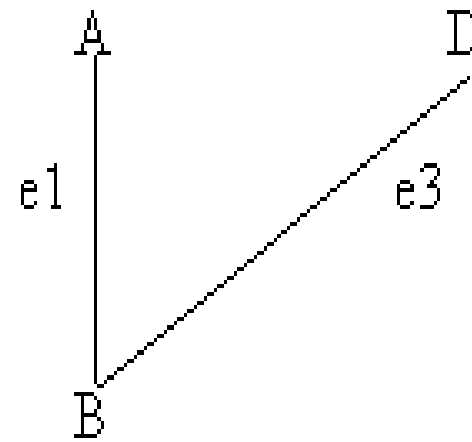
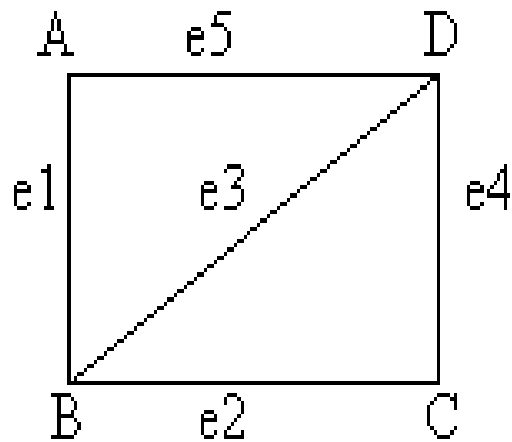


Suatu Graph yang tidak mengandung ruas sejajar maupun self-loop, sering disebut juga sebagai **Graph sederhana** atau **simple Graph**.

Suatu Graph $G'(E', V')$ disebut Sub Graph dari $G(E, V)$, bila E' himpunan bagian dari E dan V' himpunan bagian dari V .

Jika E' mengandung semua ruas dari E yang titik ujungnya di V' , maka G' disebut Subgraph yang direntang oleh V' (Spanning Subgraph).

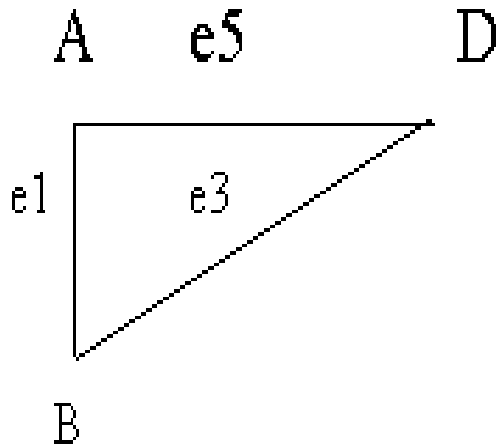
Contoh Sub Graph:



G' Subgraph dari G (namun bukan dibentuk oleh $V' = \{A, B, D\}$).

Contoh Spanning Sub Graph

G' Subgraph yang dibentuk oleh $V' = \{A, B, D\}$

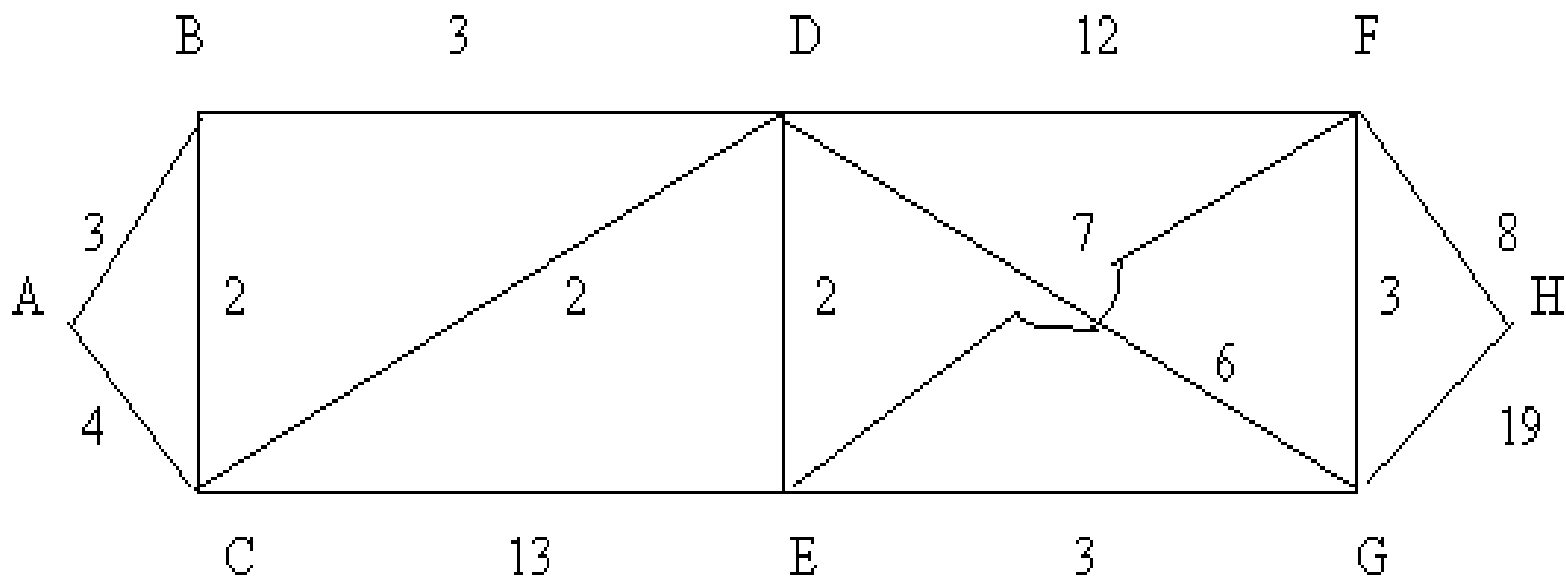


GRAPH BERLABEL

Graph G disebut berlabel jika ruas dan atau simpulnya dikaitkan dengan suatu besaran tertentu. Khususnya jika setiap Ruas e dari G dikaitkan dengan suatu bilangan non negatif $d(e)$, maka $d(e)$ disebut bobot atau panjang dari ruas e .

Contoh :

Gambar berikut ini menyajikan hubungan antar kota.
 Disini simpul menyatakan kota dan label $d(e)$ menyatakan jarak antara dua kota.

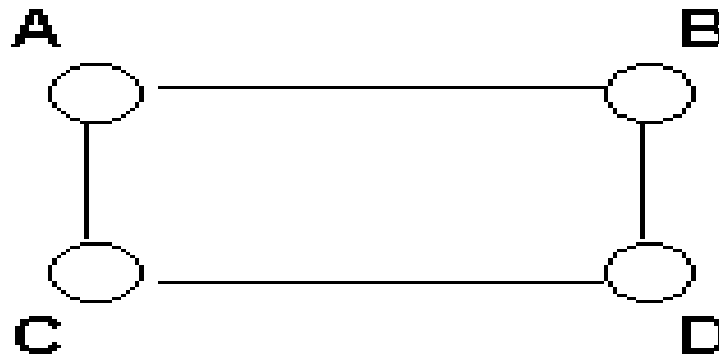


DERAJAT GRAPH

Derajat simpul V , ditulis $d(v)$ adalah banyaknya ruas yang menghubungkan v . Karena setiap ruas dihitung dua kali ketika menentukan derajat suatu Graph, maka :

Jumlah derajat semua simpul suatu Graph (derajat) = dua kali banyaknya ruas Graph (Size) Atau dapat dituliskan :

$$\text{Derajat Graph} = 2 \times \text{Size}$$



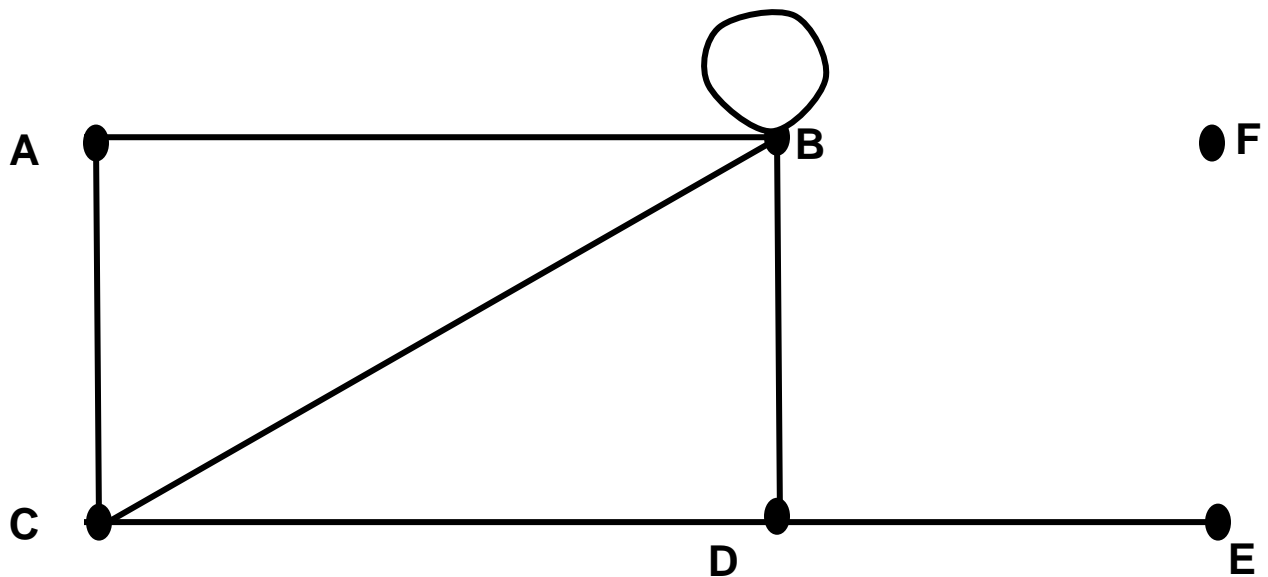
Pada gambar diatas Jumlah Semua Simpul = 4, maka
Jumlah Derajat Semua Simpul = 8

Jika Derajat masing-masing simpul pada Graph berjumlah
Genap maka Graph tersebut disebut **EULER Graph**

Suatu simpul disebut genap/ganjil tergantung apakah derajat simpul tersebut genap/ganjil.

Kalau terdapat self-loop, maka self-loop dihitung 2 kali pada derajat simpul.

Contoh :



Pada gambar diatas, banyak ruas/size = 7, sedangkan derajat masing-masing simpul adalah :

$$d(A) = 2$$

$$d(B) = 5$$

$$d(C) = 3$$

$$d(D) = 3$$

$$d(E) = 1$$

$$d(F) = 0$$

maka, total jumlah derajat simpul adalah : 14

E disebut simpul bergantung/akhir, yaitu simpul yang berderajat satu. Sedangkan F disebut simpul terpencil, yaitu simpul yang berderajat Nol.

KETERHUBUNGAN

Walk atau perjalanan dalam Graph G adalah barisan simpul dan ruas berganti-ganti : $V_1, e_1, V_2, e_2, \dots, e_{n-1}, V_n$

Disini ruas e_i menghubungkan simpul V_i dan V_{i+1} .

Banyaknya ruas disebut Panjang Walk. Walk dapat ditulis lebih singkat dengan hanya menulis deretan ruas :

e_1, e_2, \dots, e_{n-1} atau deretan simpul : $V_1, V_2, \dots, V_{n-1}, V_n$

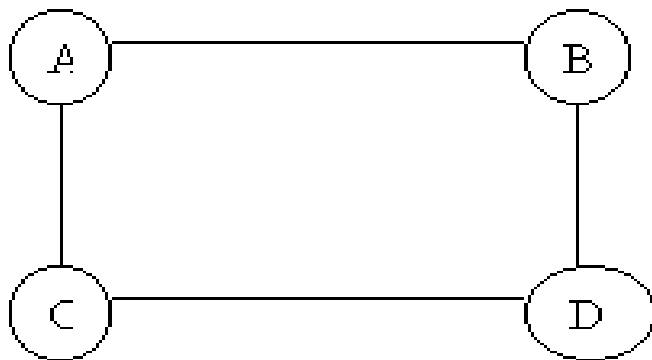
dimana : V_1 = simpul awal

V_n = simpul akhir.

Walk disebut tertutup bila $V_1 = V_n$

Istilah Pada Graph

1. Walk disebut **tertutup**, yang menghubungkan V_1 dan V_n , yaitu Setiap Ruas menghubungkan Simpul Awal dan Akhir



2. Trail adalah Walk dengan semua ruas dalam barisan adalah berbeda.
3. Path atau Jalur adalah Walk yang semua simpul dalam barisan adalah berbeda. Jadi suatu Path pastilah sebuah Trail.

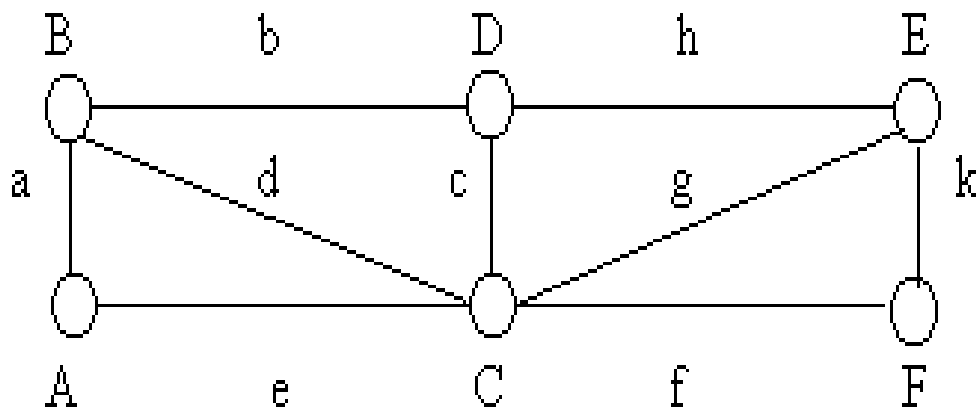


Graph merupakan Walk Terbuka, karena tidak ada ruas yang menghubungkan Simpul U dan T.

Merupakan suatu Path atau Trail terbuka dengan derajat setiap simpulnya = 2, kecuali simpul awal U dan simpul akhir T berderajat = 1.

1. Cycle atau sirkuit adalah suatu Trail tertutup dengan derajat setiap simpul = 2. Cycle dengan panjang k disebut dengan k -cycle. Demikian pula Jalur dengan panjang k disebut k -jalur.

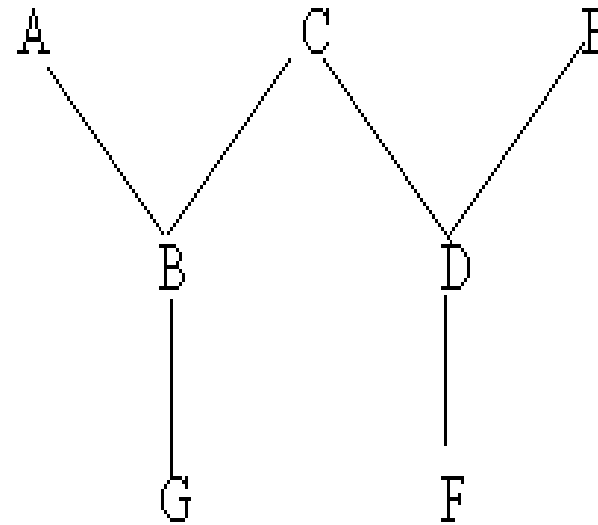
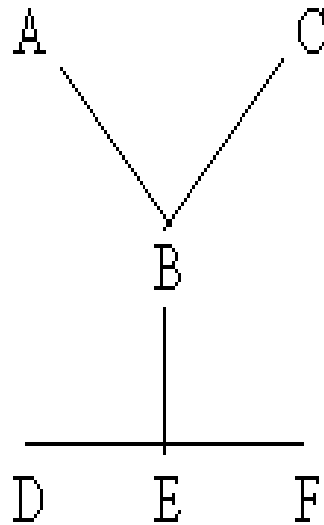
Contoh :



- Barisan ruas a,b,c,d,b,c,g,h adalah Walk bukan Trail (karena ruas b dua kali muncul).
- Barisan simpul A, B, E, F bukan Walk (karena tdk ada ruas yang menghubungkan simpul B ke F).
- Barisan simpul A, B, C, D, E, C, F adalah Trail bukan Jalur/Path (karena c dua kali muncul)
- Barisan ruas a, d, g, k adalah Jalur/Path karena menghubungkan A dengan F
- Ruas a, b, h, g, e, a, adalah Cycle.

Graph yang tidak mengandung Cycle disebut Acyclic.
Contoh dari Graph Acyclic adalah pohon atau Tree.

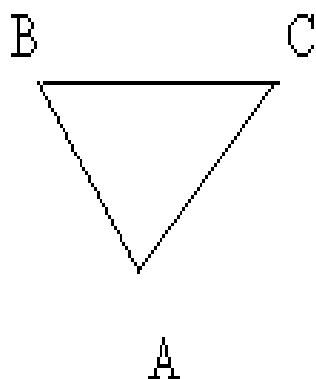
Contoh dari Acyclic



Suatu Graph G disebut terhubung jika untuk setiap 2 simpul dari Graph terdapat jalur yang menghubungkan 2 simpul tersebut.

Subgraph yang terhubung pada suatu Graph disebut komponen dari G bila Subgraph tersebut tidak terkandung dalam Subgraph terhubung lain yang lebih besar.

Contoh :



Terlihat misalnya antara D dan A
Tak ada jalur.

GRAPH TERARAH (DIRECTED GRAPH / DIGRAPH)

Graph terarah adalah Graph yang dapat menghubungkan V_1 ke V_2 saja (1 arah).

Maksimum jumlah busur dari n simpul adalah : $n (n - 1)$

Suatu Graph Berarah (Directed Graph) D terdiri atas 2 himpunan :

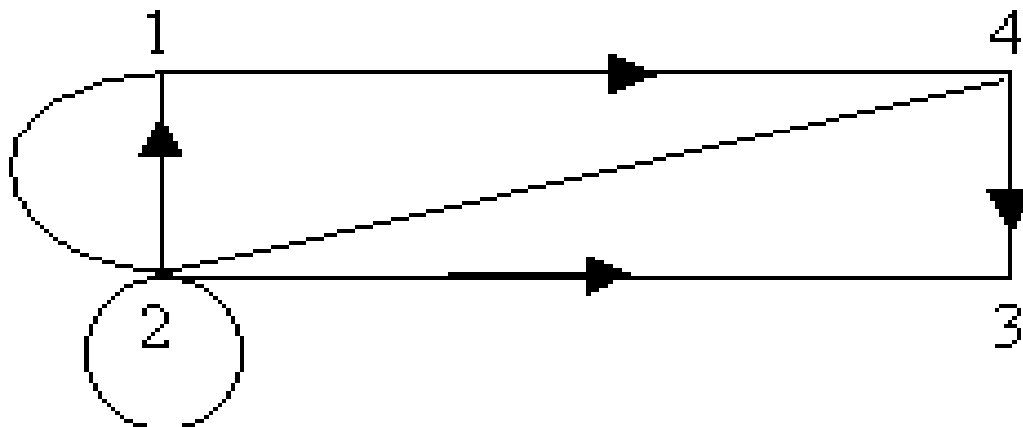
- 1) Himpunan V , anggotanya disebut simpul.
- 2) Himpunan A , merupakan himpunan pasangan terurut, yang disebut ruas berarah atau arkus.

Contoh :

Gambar dibawah ini adalah sebuah Graph Berarah $D(V,A)$ dengan :

1. V mengandung 4 simpul, yaitu 1, 2, 3 dan 4
2. A mengandung 7 arkus, yaitu $(1,4)$, $(2,1)$, $(2,1)$, $(4,2)$, $(2,3)$, $(4,3)$ dan $(2,2)$

Arkus $(2,2)$ disebut gelung (self-loop), sedangkan arkus $(2,1)$ muncul lebih dari satu kali, disebut arkus sejajar atau arkus berganda.



Bila arkus suatu Graph Berarah menyatakan suatu bobot, maka Graph Berarah tersebut dinamakan jaringan / Network. Biasanya digunakan untuk menggambarkan situasi dinamis.

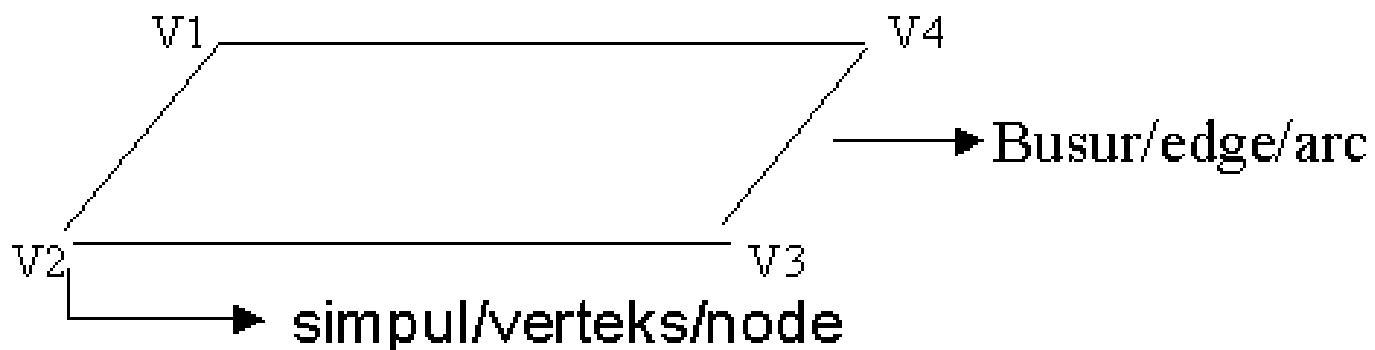
Bila V' himpunan bagian dari V serta A' himpunan bagian dari A , dengan titik ujung anggota A' terletak di dalam V' , maka dikatakan bahwa $D'(V', A')$ adalah Graph bagian (Subgraph) dari $D(V, A)$.

Bila A' mengandung semua arkus anggota A yang titik ujungnya anggota V' , maka dikatakan bahwa $D'(V', A')$ adalah Graph Bagian yang dibentuk atau direntang oleh V' .

Undirected Graph

GRAPH TAK TERARAH (UNDIRECTED GRAPH)

Graph Tak Terarah adalah Graph yang menghubungkan 2 verteks $V1$ ke $V2$ dan $V2$ ke $V1$ (2 arah). Bila Verteks = n , maka Graph tak terarah komplit akan mempunyai busur edge sama dengan : $\boxed{n(n-1)/2}$



Yang dapat dilakukan adalah :

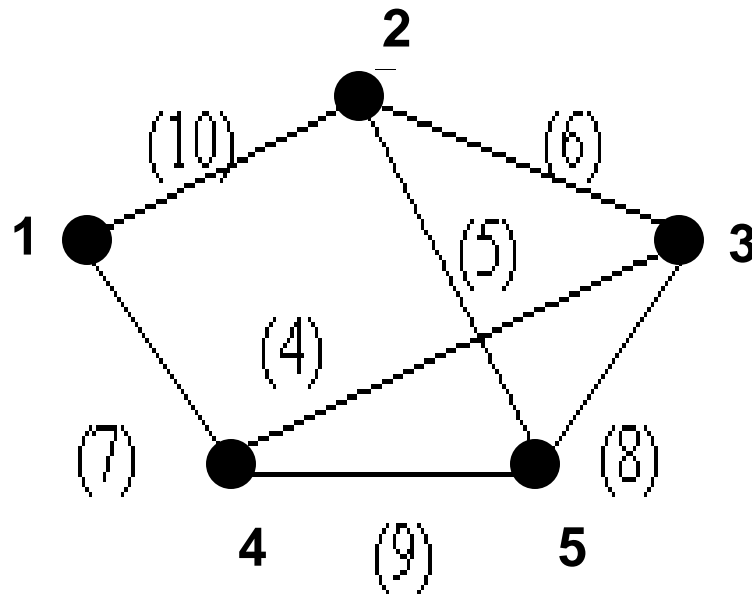
$$V1 - V2 = V2 - V1$$

$$V1 - V4 = V4 - V1$$

dan seterusnya.

Critical Path

Menggunakan Graph berbobot dan Mempunyai Arah



Simpul asal : 1

Simpul Tujuan : 5

Critical Path

	PATH	BOBOT
Alternatif :	1-->4 -->5	16
	1-->2-->5	15
	1-->2-->3-->5	24
	1-->4-->3-->5	19
	1-->2-->3-->4-->5	29
	1-->4-->3-->2-->5	22

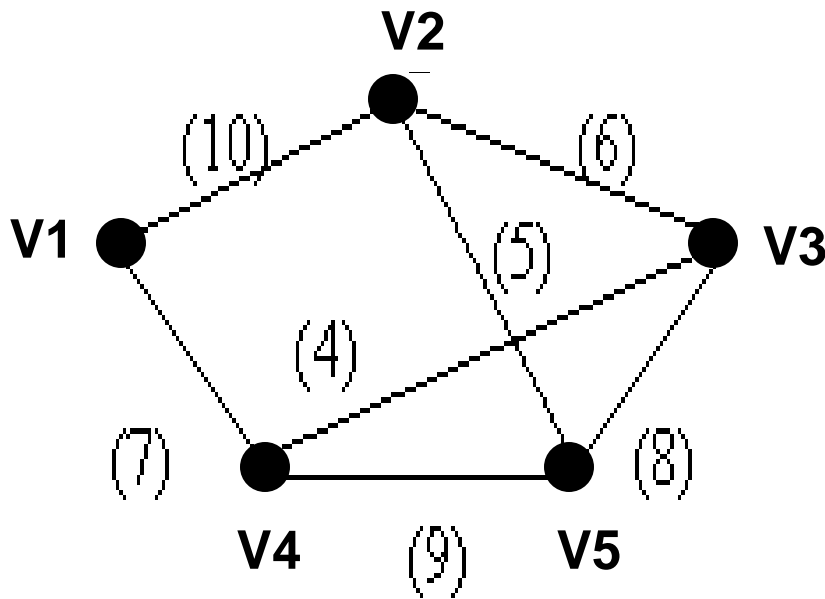
Diperoleh : Critical Path (Lintasan Kritis) = 29

Shortest Path (Lintasan Terpendek) = 15

Minimum Spanning Tree

Merupakan Spanning Tree yang mempunyai Bobot dan tidak mempunyai arah dengan hasil penjumlahan bobotnya adalah minimum.

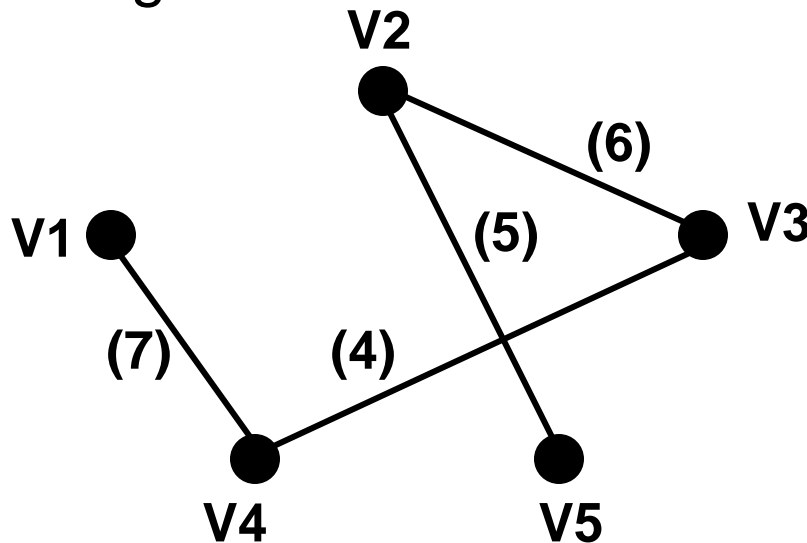
Lihat gambar Graph G berikut :



Minimum Spanning Tree

Langkah yang dilakukan untuk membentuk minimum spanning tree adalah :

Bentuk kembali semua simpul tetapi tanpa ruas.
Gambar dan telusuri ruas dengan bobot paling kecil, seterusnya (secara ascending) hingga semua simpul terhubung



Total Minimum
Spanning Tree = 22

MATRIKS PENYAJIAN GRAPH

Misalnya disajikan Graph G dalam Matriks ruas B ukuran $(M \times 2)$, maka setiap baris Matriks menyatakan ruas, misalnya baris (4 7) menyatakan ada ruas menghubungkan simpul 4 dan 7.

Matriks Adjacency dari Graph G, yaitu Matriks yang menghubungkan Vertex dengan Vertex, tanpa ruas sejajar adalah Matriks A berukuran $(N \times N)$ yang bersifat :

$$a_{ij} = \begin{cases} 1, & \text{bila ada ruas } (V_i, V_j) \\ 0, & \text{bila dalam hal lain.} \end{cases}$$

Matriks Adjacency merupakan matriks simetri.

Untuk Graph dengan ruas sejajar, Matriks Adjacency didefinisikan sebagai berikut :

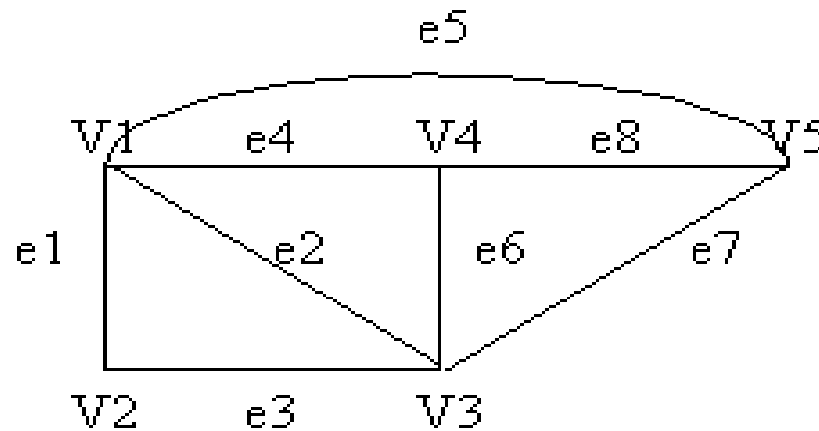
$$a_{ij} = \begin{cases} P, & \text{bila ada } p \text{ buah ruas menghubungkan} \\ & (V_i, V_j) (p > 0) \\ 0, & \text{bila dalam hal lain.} \end{cases}$$

Matriks Incidence dari Graph G, yaitu Matriks yang menghubungkan Vertex dengan Edge, tanpa self-loop didefinisikan sebagai Matriks M berukuran (NXM) sebagai berikut :

$$m_{ij} = \begin{cases} 1, & \text{bila ada ruas } e_j \text{ berujung di simpul } V_i \\ 0, & \text{dalam hal lain.} \end{cases}$$

Matriks Adjacency

Contoh :



Matriks Ruas

1	2
1	3
1	4
1	5
2	3
3	4
3	5
4	5

atau :

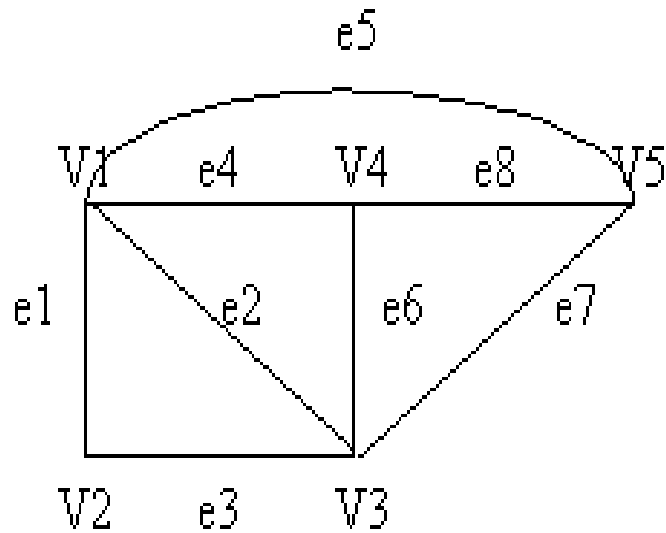
1	1	1	1	2	3	3	4
2	3	4	5	3	4	5	5

atau secara pasangan

$\{(1,2),(1,3),(1,4),(1,5),(2,3),(3,4),(3,5),(4,5)\}$

Matriks Adjacency

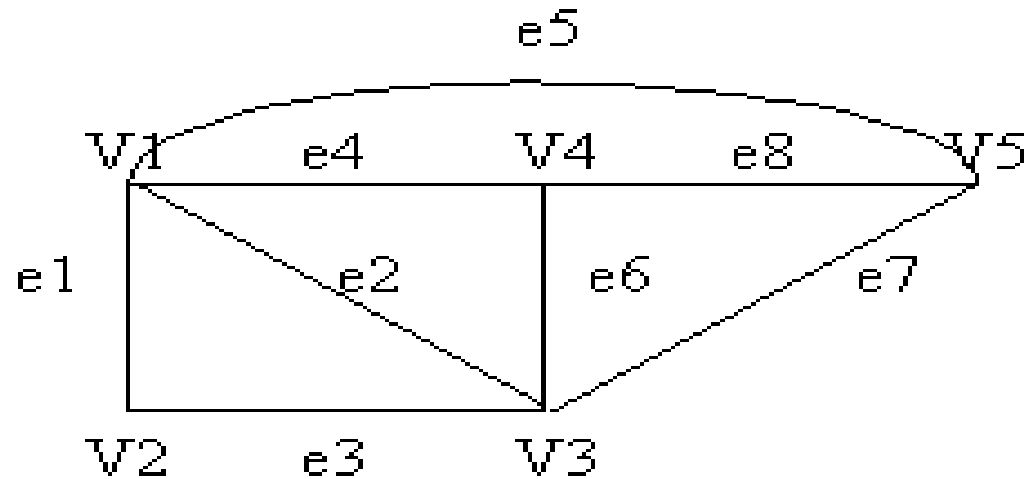
Contoh :



Matriks Adjacency

	V1	V2	V3	V4	V5
V1	0	1	1	1	1
V2	1	0	1	0	0
V3	1	1	0	1	1
V4	1	0	1	0	1
V5	1	0	1	1	0

Contoh :



Matriks Incidence

	V1	V2	V3	V4	V5	
		e1	e2	e3	e4	e5
V1	1	1	0	1	1	0
V2	1	0	1	0	0	0
V3	0	1	1	0	0	1
V4	0	0	0	1	0	1
V5	0	0	0	0	0	1

GRAPH TERARAH (DIRECTED GRAPH / DIGRAPH)

Graph terarah adalah Graph yang dapat menghubungkan V_1 ke V_2 saja (1 arah).

Maksimum jumlah busur dari n simpul adalah : $n(n - 1)$

Suatu Graph Berarah (Directed Graph) D terdiri atas 2 himpunan :

- 1) Himpunan V , anggotanya disebut simpul.
- 2) Himpunan A , merupakan himpunan pasangan terurut, yang disebut ruas berarah atau arkus.

Graph

PENELUSURAN GRAPH

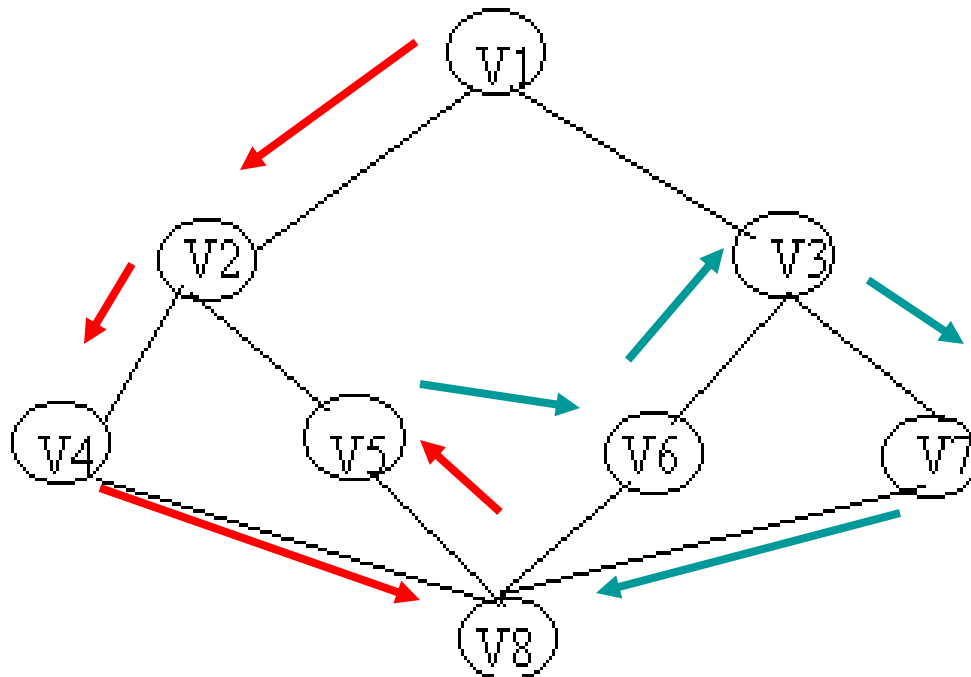
Dapat dilakukan dengan 2 cara , yaitu :

1. Depth First Search (DFS)
2. Breadth First Search (BFS)

1. Depth First Search (DFS)

Penelusuran dengan DFS pada Graph Tak Berarah dengan melakukan pengecekan pada Node dengan kedalaman pertama dari Node yang ditinjau.

Depth First Search



Karena V8 sudah dilewati setelah penelusuran ke V4, maka penelusuran yang berikutnya dianggap tidak dilewati lagi

Klik Animasi

Dari gambar diatas dapat terbentuk matriks sbb :

	V1	V2	V3	V4	V5	V6	V7	V8
V1	0	1	0	0	0	0	0	0
V2	0	0	0	1	0	0	0	0
V3	0	0	0	0	0	0	1	0
V4	0	0	0	0	0	0	0	1
V5	0	0	0	0	0	1	0	0
V6	0	0	1	0	0	0	0	0
V7	0	0	0	0	0	0	0	1
V8	0	0	0	0	1	0	0	0

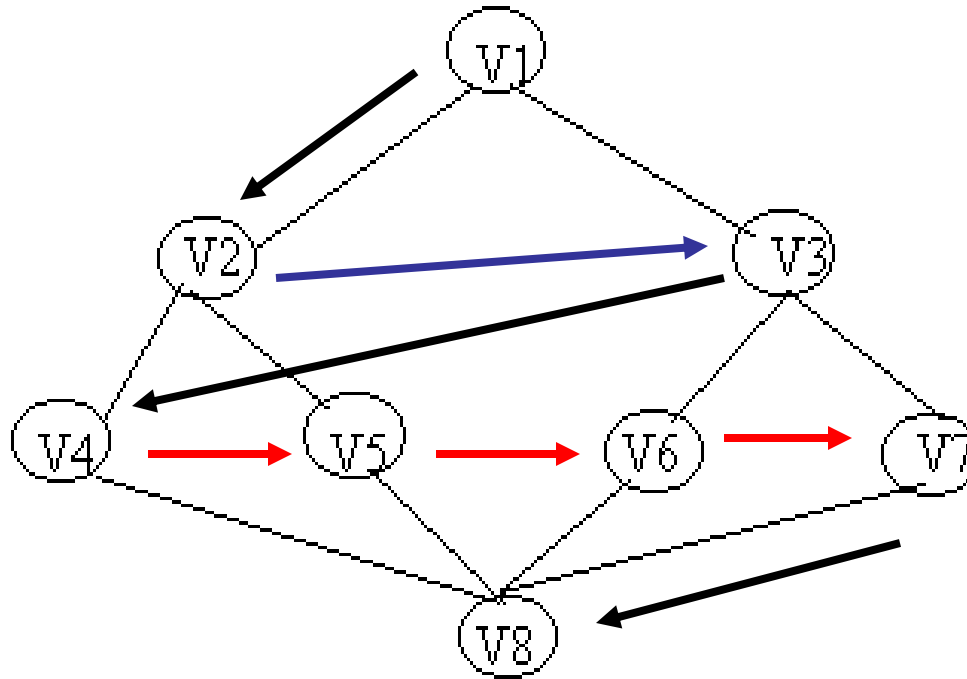
Dari Matriks diatas, akan diperoleh urutan sbb:

V1 -> V2 -> V4 -> V8 -> V5 -> V6 -> V3 -> V7

2. Breadth First Search (BFS).

Berbeda dengan cara DFS, dengan BFS penelusuran akan diawasi dari Node-1, kemudian melebar pada Adjacent Node dari Node-1 dan diteruskan pada Node-2, Node- 3 dan seterusnya.

Breadth First Search



Dari gambar di atas akan diperoleh urutan :
V1 , **V2** ---> **V3** , **V4** ---> **V5** ---> **V6** ---> **V7** , **V8**

Klik Animasi

Pertemuan 13-14

PRESENTASI PROJECT