



# Tree

Tim Ajar Algoritma dan Struktur Data  
Genap 2023/2024



# Tujuan

## STRUKTUR DATA TIDAK LINEAR

- Memahami definisi dan terminologi Tree
- Memahami aplikasi / penerapan Tree
- Memahami bagaimana menelusuri (traverse) Tree

# Outline

- Tree
- Binary Tree
- Representasi Binary Tree
- Binary Search Tree
- Binary Tree Traversal
- Operasi Pada Binary Search Tree

# Pendahuluan

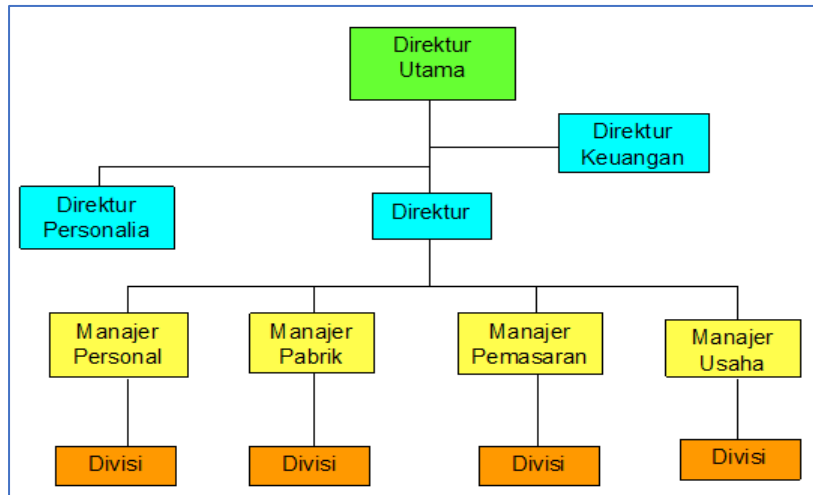
- Ingat, List, Stack dan Queue merupakan struktur data yang bersifat linear (Linear List)
- Linear List digunakan untuk data yang terurut secara serial  
    contoh: antrian, nama hari dalam minggu, nama bulan dalam tahun, dll
- *Tree* merupakan salah satu bentuk implementasi banyak *linked list* (*double linked list*) yang biasanya digunakan untuk menggambarkan hubungan yang bersifat hirarkis antara elemen-elemen yang ada
- ▶ *Tree* → digunakan untuk data yang terurut secara hirarki (one to many).
  - **Contoh :**
    - Pohon silsilah keluarga
    - Struktur organisasi suatu instansi/perusahaan
    - Diagram pertandingan sistem gugur
    - Table of content / daftar isi sebuah buku
    - File system dari DOS/Windows atau Unix

Beringkat atau bercabang  
antar elemen, data yang  
terurut secara hierarki

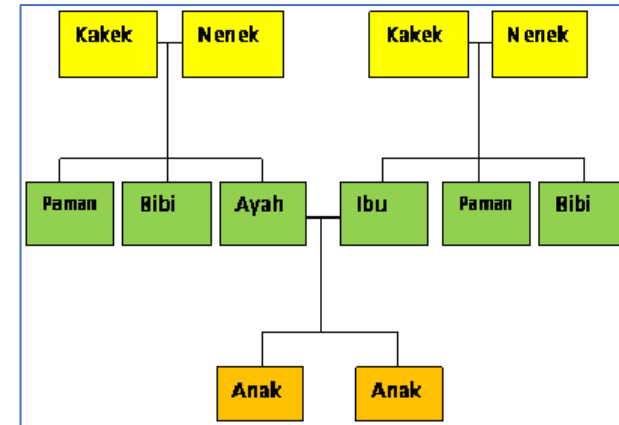
# Contoh Tree



Diagram Pertandingan Sistem Gugur



Struktur Organisasi



Silsilah Keluarga

# Contoh Tree (2)

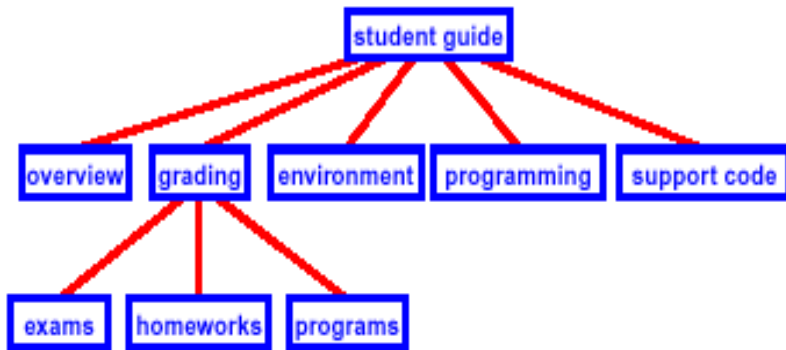
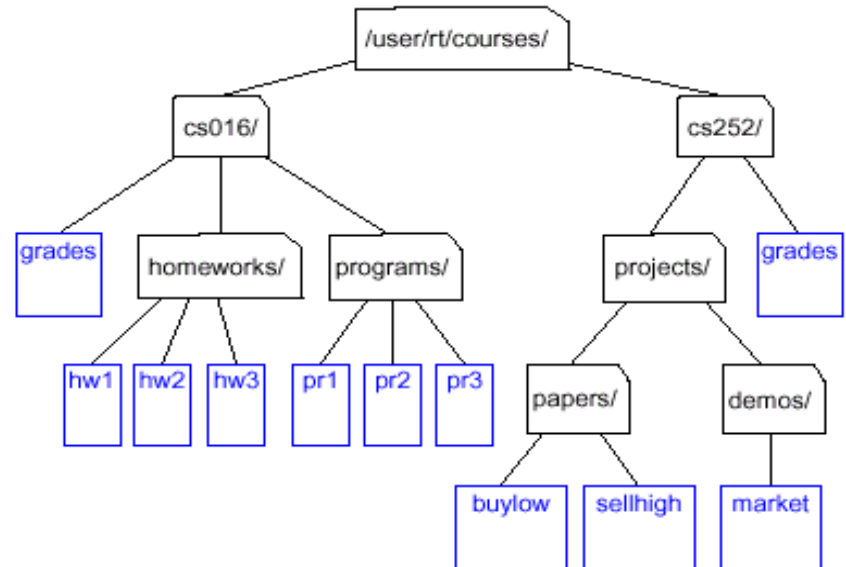


Table of Content/ Daftar isi sebuah buku



File System dari DOS/Windows atau Unix



# Tree

# Definisi *Tree*

1 elemen bisa terhubung ke banyak elemen

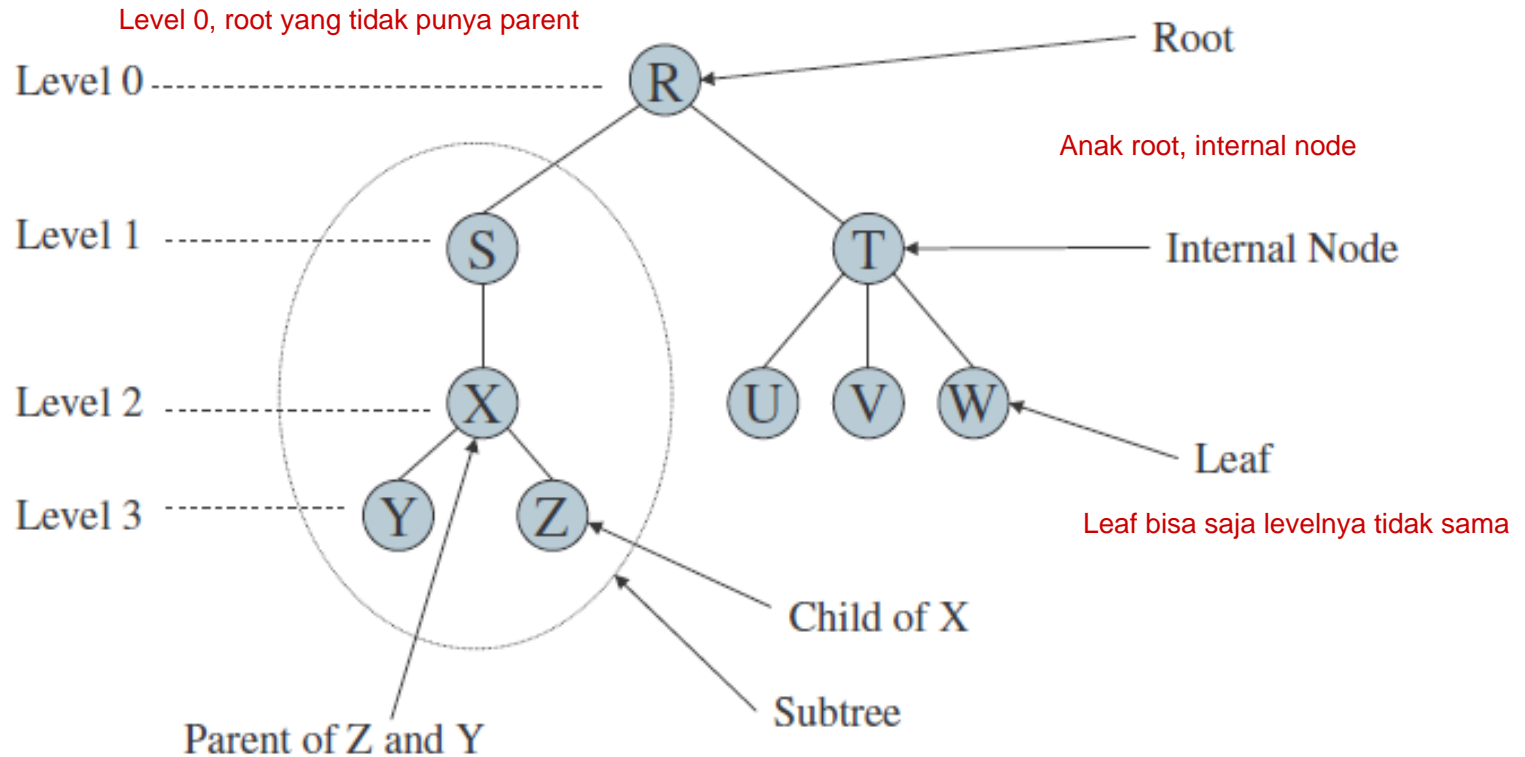
- *Tree* adalah kumpulan elemen yang saling terhubung secara hirarki (*one to many*).
- Elemen pada tree disebut *node*.
- Garis yang menghubungkan tree disebut edge (terkadang disebut *line*)

## Aturan :

- Sebuah *node* hanya boleh memiliki satu induk/*parent*. Kecuali *root*, tidak memiliki induk/*parent*.
- Setiap *node* dapat memiliki nol atau banyak anak/*child* (*one to many*).
- *Node* yang tidak memiliki anak/*child* disebut daun/*leaf*.



# Tree Anatomy

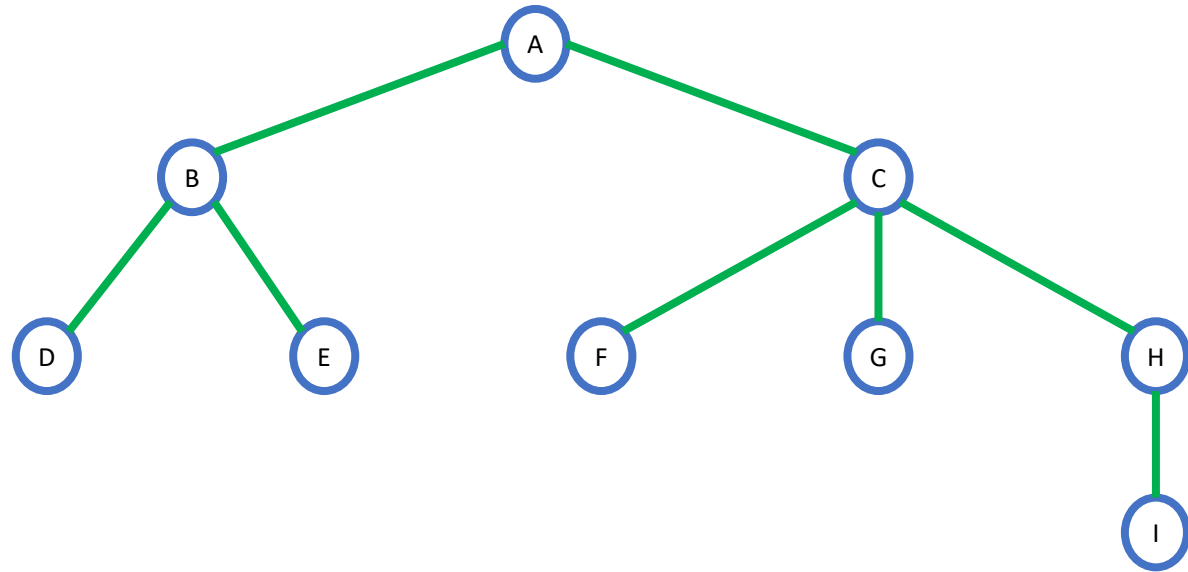




# Istilah/ Terminologi dalam *Tree*

- **Node** : sebuah emelen dalam sebuah **tree**; berisi sebuah informasi.
- **Predecessor** : *node* yang berada di atas node tertentu
- **Successor** : *node* yang berada di bawah node tertentu.
- **Ancestor** : seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
- **Descendant** : Seluruh node yang terletak sesudah node tertentu dan terletak pada jalur yang sama.
- **Parent** : Predecessor satu level di atas satu node.
- **Child** : Successor satu level di bawah satu node.
- **Sibling** : Node yang memiliki parent yang sama dengan satu node.
- **Subtree** : Bagian dari tree yang berupa suatu node beserta descendantnya.
- **Size** : Banyaknya node dalam suatu tree.
- **Height** : Banyaknya tingkat/level dalam suatu tree.
- **Root** (Akar) : Node khusus dalam tree yang tidak memiliki predecessor.
- **Leaf** (Daun) : Node-node dalam tree yang tidak memiliki successor.
- **Degree** (Derajat) : Banyaknya child yang dimiliki oleh suatu node

# Contoh *Tree*

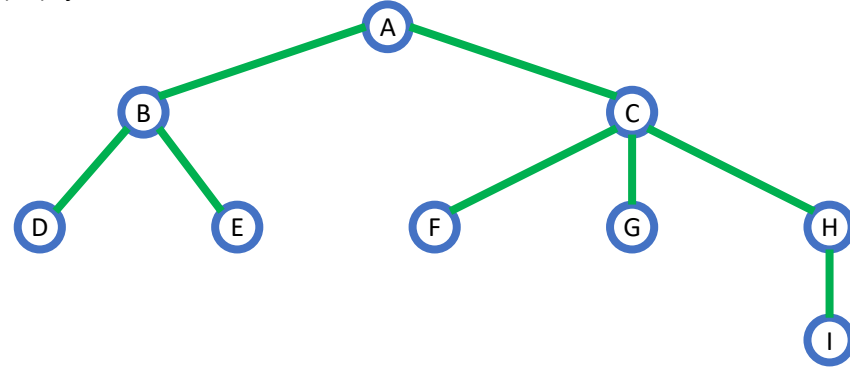


Latihan, bagian mana sajakah (node) dari tree di atas, yang sesuai dengan terminology dalam tree?

# Jawab



- **Node** : Terdapat 9 *Node* dalam contoh *Tree*, yaitu node {A, B, C, D, E, F, G, H, I}
- **Predecessor** : Node B merupakan predecessor dari Node {D, E}
- **Successor** : node {D, E} merupakan successor dari node B
- **Ancestor** : node {C, A} merupakan ancestor dari node F
- **Descendant** : node {D, E} merupakan descendant dari node B
- **Parent** : node C adalah parent dari node {F, G, H}
- **Child** : node {B, C} adalah child dari node A
- **Sibling** : node {F, G, H} adalah sibling, memiliki parent yang sama yaitu node C
- **Subtree** : terdapat 2 subtree, yaitu subtree dari node {B, D, E} dan dari node {C, F, G, H, I}
- **Size** : dalam tree terdapat 9 node.
- **Height** : tingkatan tree adalah 4
- **Root** : node A adalah root dari tree.
- **Leaf** : node yang menjadi leaf adalah node {D, E, F, G, I}
- **Degree** : node B memiliki derajat 2 {D, E} dan node C memiliki derajat 3 {F, G, H}



# Tree Facts

- Setiap *node*, kecuali *root*, memiliki *HANYA* satu *parent*.
- *Subtree* merupakan kumpulan child-child dari sebuah *node*, yang membentuk struktur tree yang lebih kecil



# Binary Tree

# Binary Tree (Pohon Biner)

- Dalam tree terdapat jenis-jenis tree yang memiliki sifat khusus, diantaranya adalah **binary tree**.
- *Binary Tree* adalah sebuah pengorganisasian secara hirarki dari beberapa buah *node*, dimana untuk setiap *node* HANYA memiliki maksimal 2 child.
- Secara khusus 2 child yang dimaksud, dinamakan kiri (*left-child*) dan kanan (*right-child*).

# Jenis *Binary Tree*

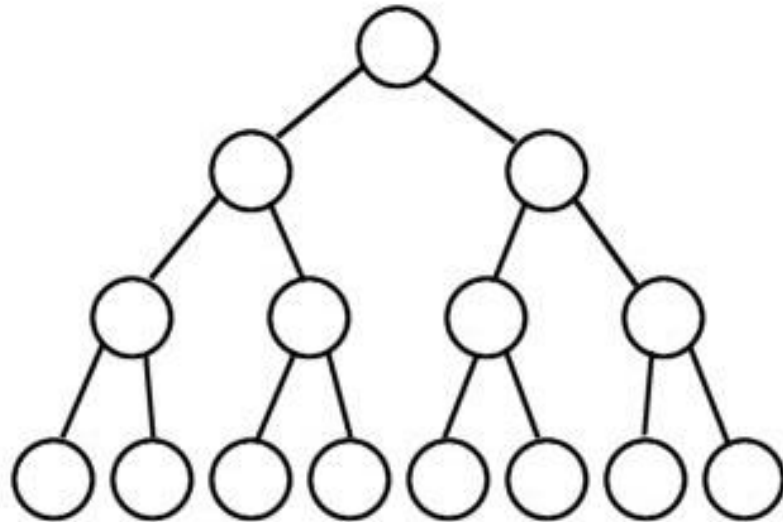
Terdapat beberapa jenis *binary tree* didasarkan pada struktur susunan node-node di dalamnya antara lain:

- *Full Binary Tree*
- *Strict Binary Tree*
- *Complete Binary Tree*
- *Incomplete Binary Tree*
- *Skewed Binary Tree*



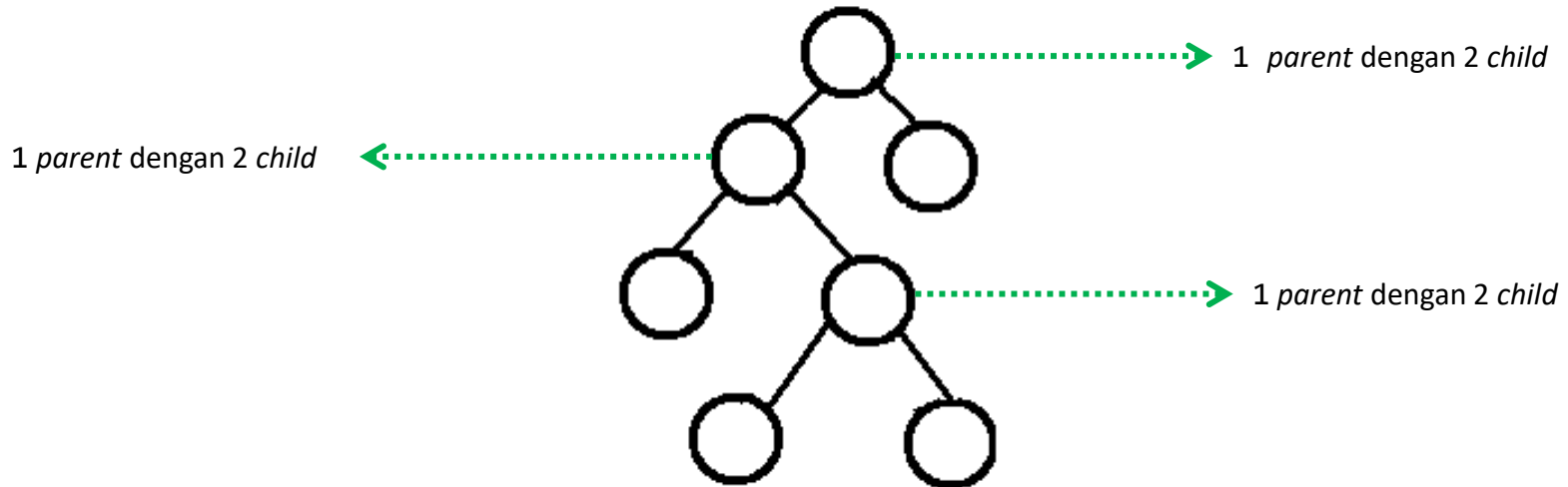
# Full Binary Tree

- Semua *node* (kecuali *leaf*) memiliki 2 anak dan tiap cabang memiliki panjang ruas yang sama. Atau, tiap subtree memiliki panjang *path* yang sama
- Disebut juga *maximum binary tree* ataupun *perfect binary tree*.



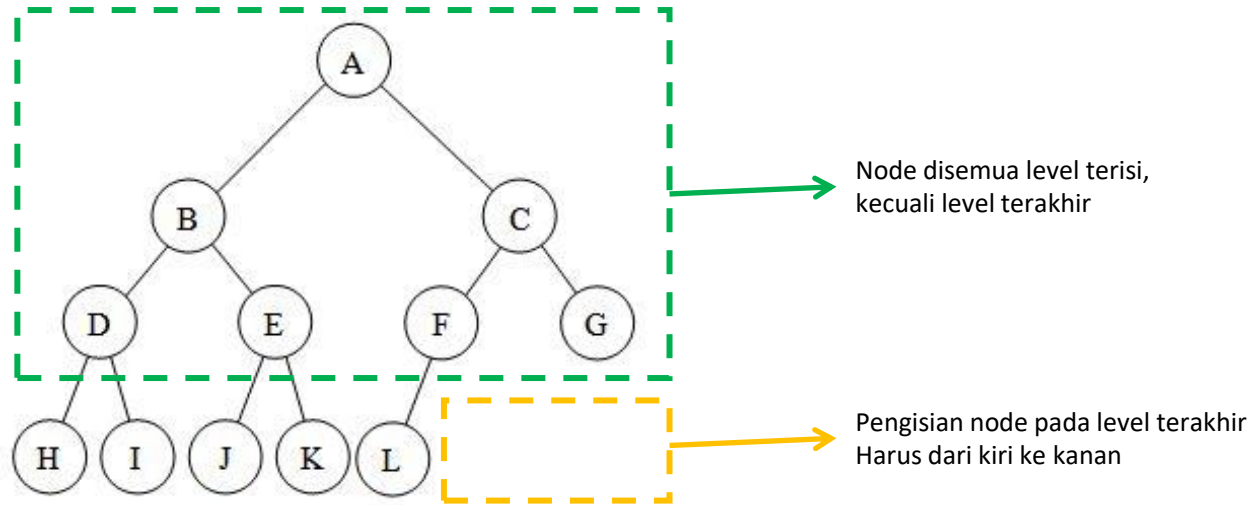
# Strict Binary Tree

- Setiap *parent node* HARUS memiliki 2 child, tidak ada *parent node* yang HANYA memiliki satu child.
- Tiap subtree tidak diharuskan memiliki panjang path yang sama dengan subtree yang lain seperti pada full binary tree.



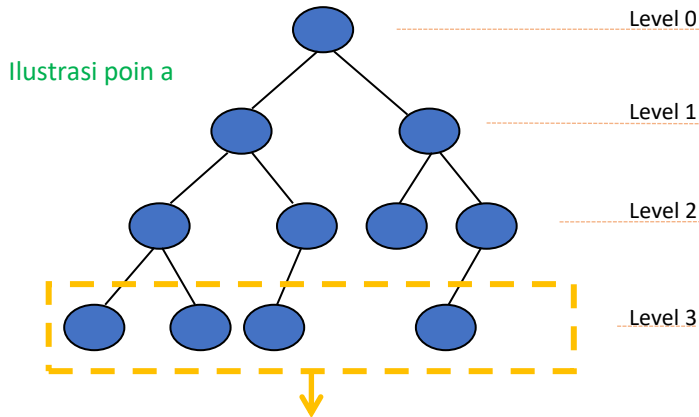
# Complete Binary Tree

- Sebuah tree dimana telah terisi node secara lengkap pada semua level, kecuali untuk level terakhir.
- Pada level terakhir, node dapat diisi secara tidak lengkap akan tetapi pengisian dari node HARUS secara terurut dari kiri ke kanan.

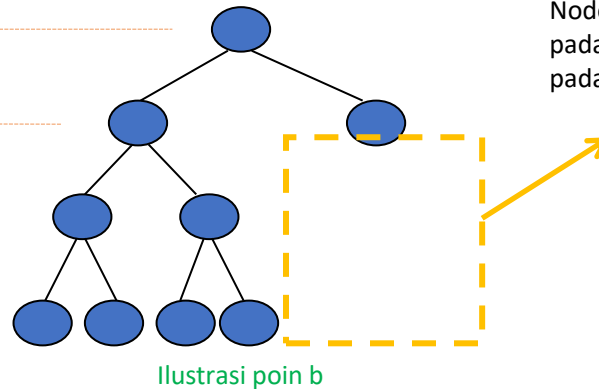


# Incomplete Binary Tree

- a. Seluruh node pada level terakhir diisi secara tidak runtut dari bagian kiri ke kanan terlebih dahulu.
- b. Terdapat node yang kosong pada tree selain diposisi level terakhir.

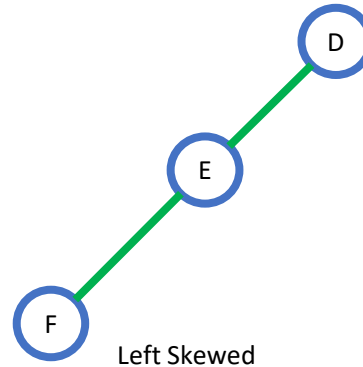
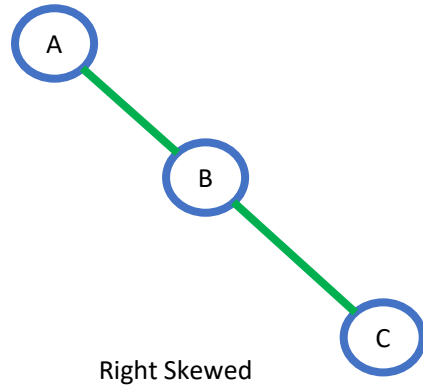


Level terakhir, node yang diisikan tidak runtut dari node paling kiri.



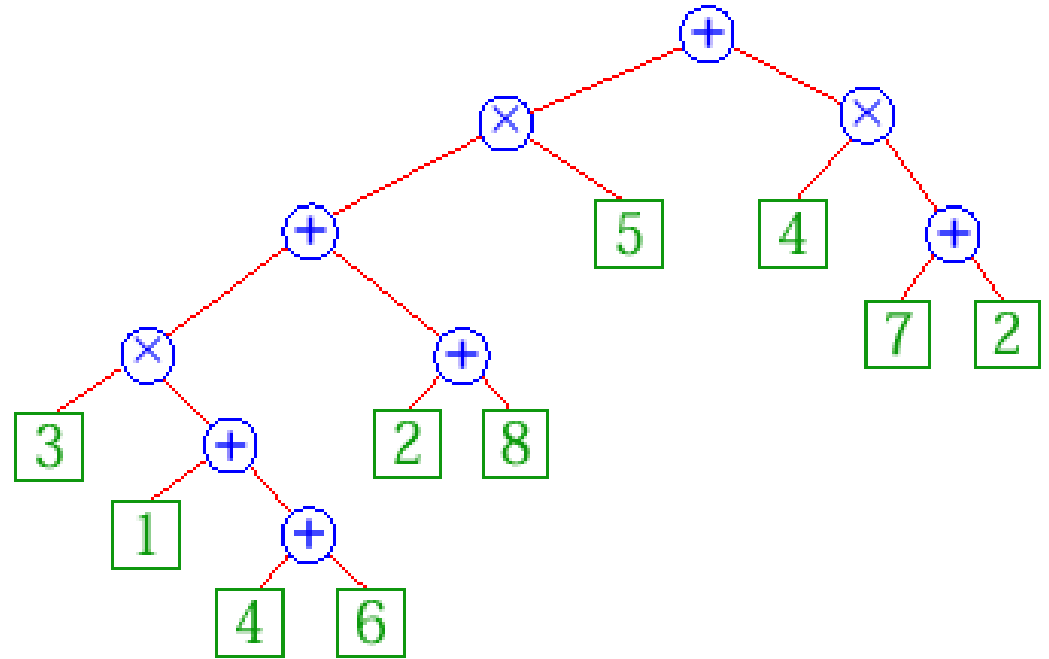
# Skewed Binary Tree

- *Binary tree* yang semua *node*-nya (kecuali *leaf*) hanya memiliki satu anak.
- Disebut juga *minimum binary tree*.



# Contoh Binary Tree

- Representasi ekspresi arithmatik



$$(((3 \times (1 + (4 + 6))) + (2 + 8)) \times 5) + (4 \times (7 + 2)))$$

# Latihan

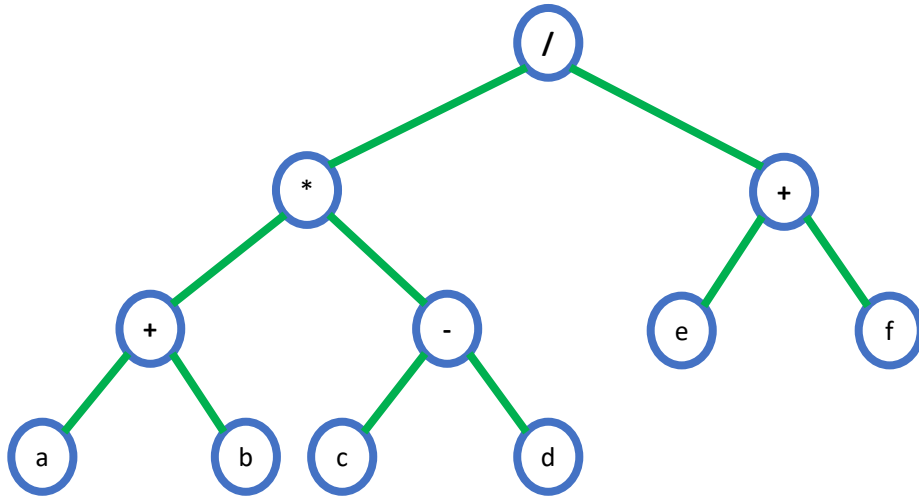
Buatlah binary tree dari expresi aritmatik berikut:

1.  $(a + b) * (c - d) / (e + f)$
2.  $(a + b) * ((b - c) + d)$

# Latihan - Jawab 1

Buatlah binary tree dari expresi aritmatik berikut:

1.  $(a + b) * (c - d) / (e + f)$

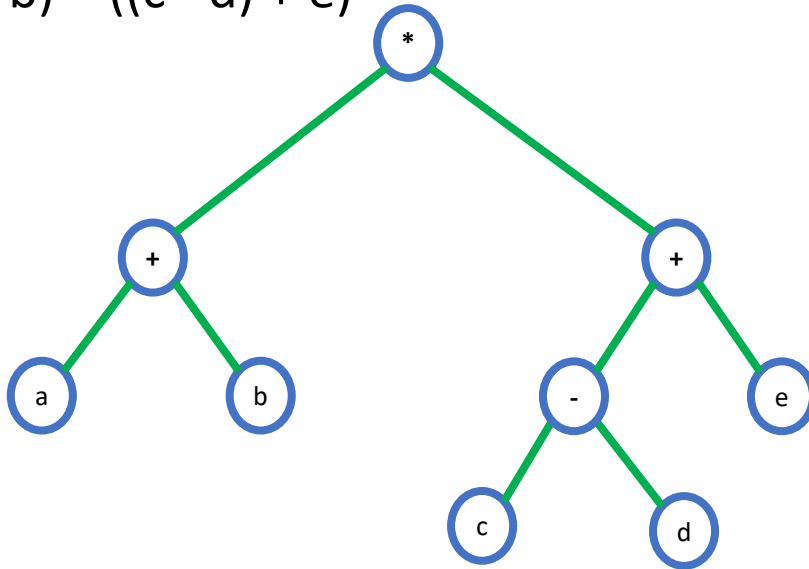




# Latihan - Jawab 2

Buatlah binary tree dari expresi aritmatik berikut:

2.  $(a + b) * ((c - d) + e)$



# Tugas Latihan 1

Buatlah binary tree dari expresi aritmatik berikut:

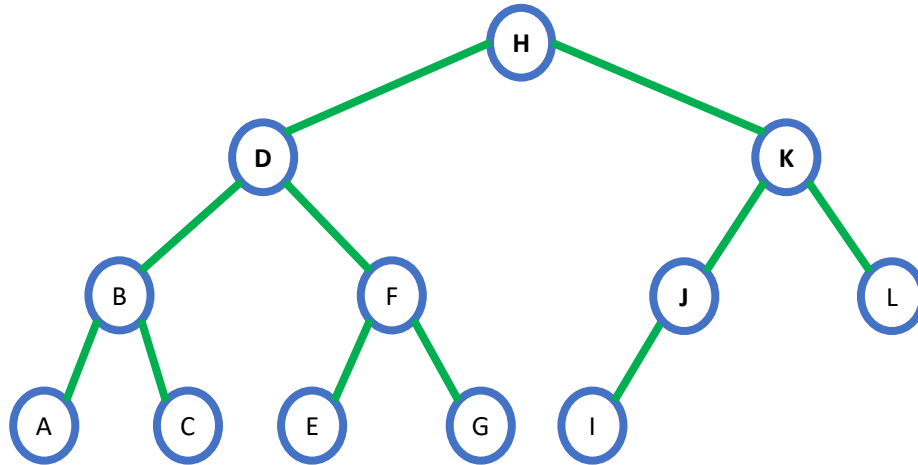
1.  $a * (b + c) / (e + (f - g))$
2.  $((a * b) * c) + (d / e) * f$



# Representasi Binary Tree

# Representasi *Binary Tree*

Binary Tree dapat direpresentasikan dengan menggunakan *array* maupun *linked list*.





# Representasi *Tree* dengan *Array*

Posisi *node* dapat ditentukan berdasarkan rumus berikut :

- Asumsi root dimulai dari indeks-0 :
  - Anak kiri dari node  $i$  berada pada indeks :  $2*i+1$
  - Anak kanan dari node  $i$  berada pada indeks :  $2*i+2$
- Asumsi root dimulai dari index 1 :
  - Anak kiri dari node  $i$  berada pada indeks :  $2*i$
  - Anak kanan dari node  $i$  berada pada indeks :  $2*i+1$

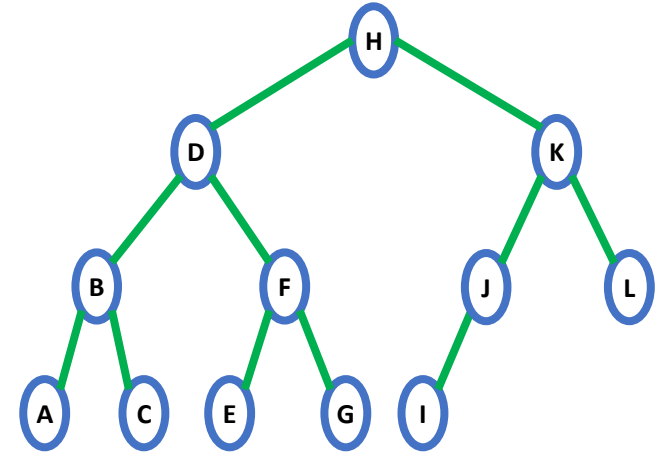
# Representasi *Tree* dengan *Array* (cont.)

- Asumsi root dimulai dari indeks-0

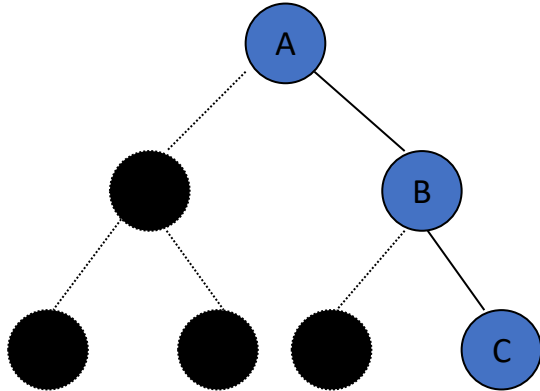
H	D	K	B	F	J	L	A	C	E	G	I
0	1	2	3	4	5	6	7	8	9	10	11

- Asumsi root dimulai dari indeks-1 :

	H	D	K	B	F	J	L	A	C	E	G	I
0	1	2	3	4	5	6	7	8	9	10	11	12



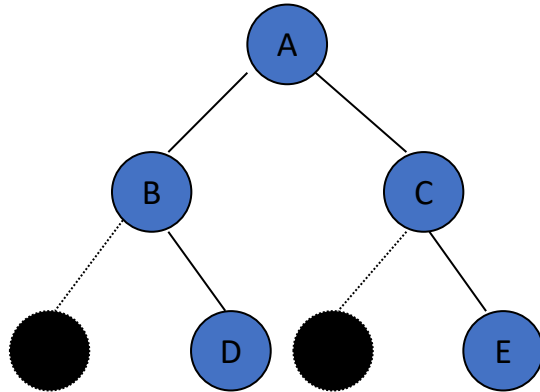
# Representasi *Tree* dengan *Array* (cont.)



- Pohon biner disamping mempunyai 3 node {A, B, C}
- Node yang berwarna hitam menggambarkan elemen yang hilang



# Representasi *Tree* dengan *Array* (cont.)



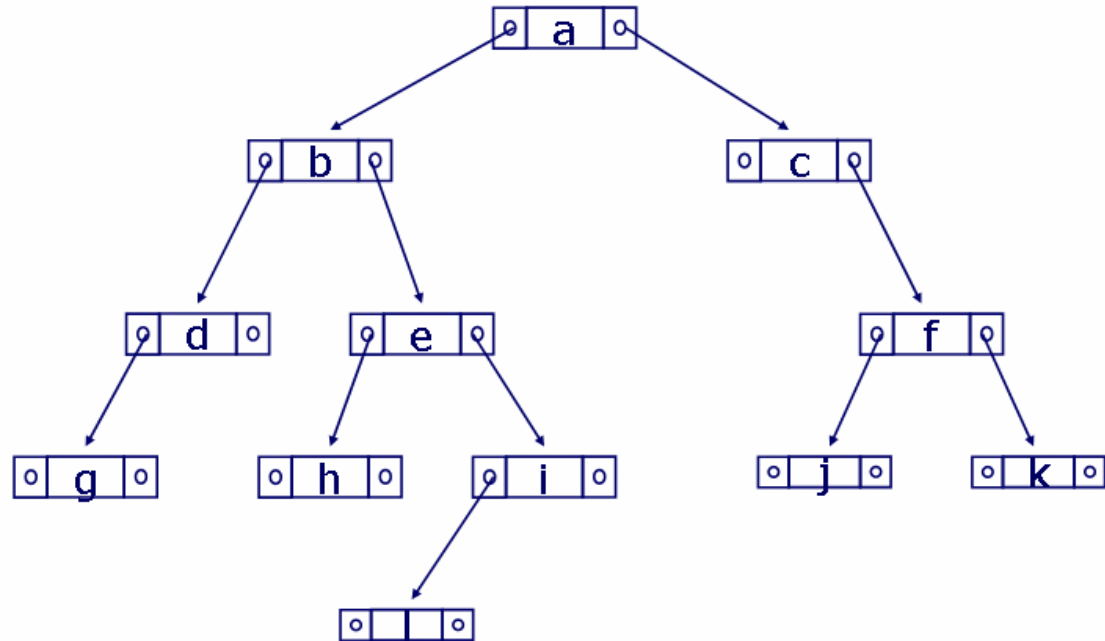
- Pohon biner di samping mempunyai 5 node {A, B, C, D, E}
- Node yang berwarna hitam menggambarkan elemen yang hilang



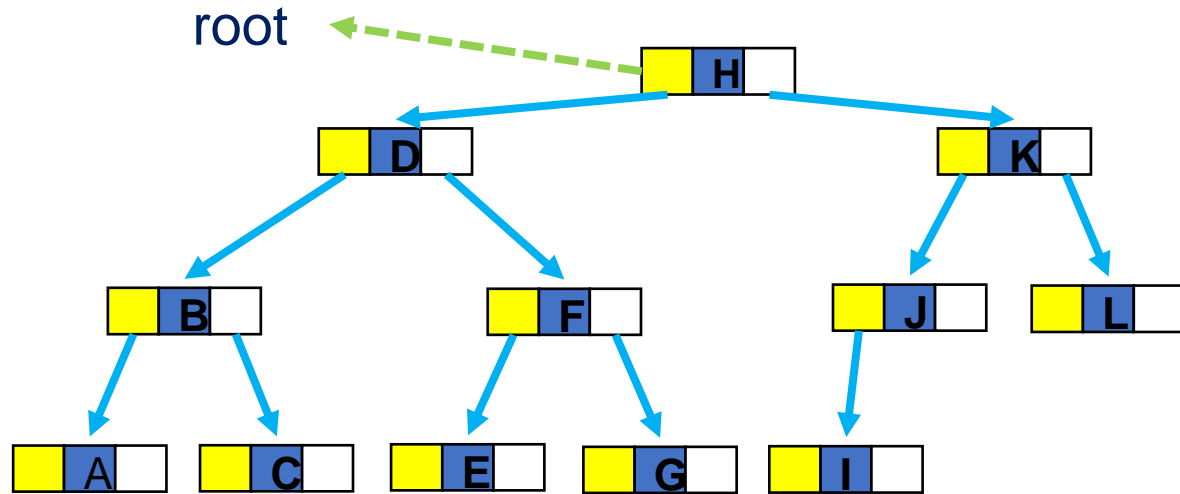
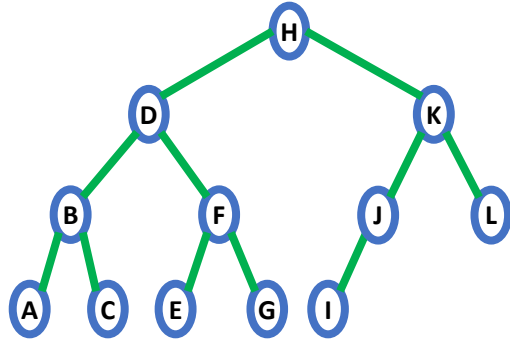


# Representasi *Tree* dengan *Linked List*

- Implementasi *binary tree* bisa dilakukan menggunakan struktur data *linked list*; masing-masing *node* terdiri atas 3 bagian yaitu
  1. Pointer Kiri
  2. Info data / elemen
  3. Pointer Kanan



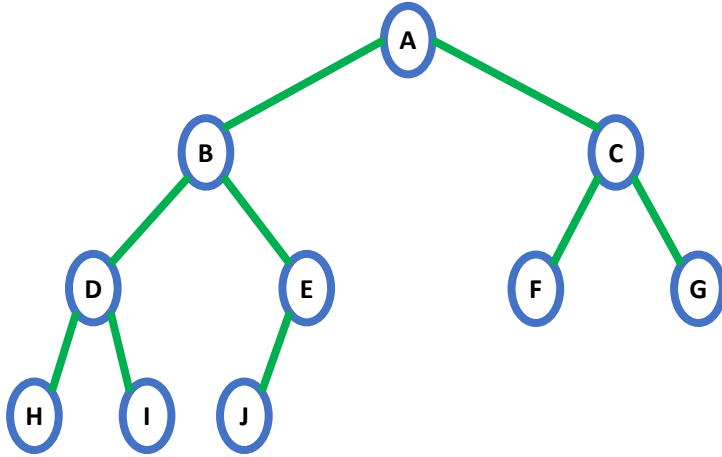
# Representasi *Tree* dengan *Linked List* (cont.)



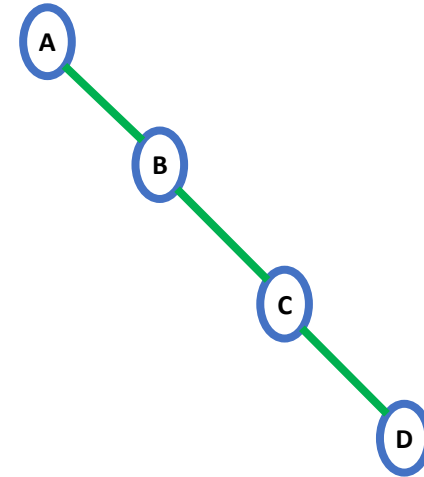
leftChild  
  element  
  rightChild

# Tugas Latihan 2

- Representasikan tree berikut dengan ilustrasi **array** dan **linked list**.



Latihan 1



Latihan 2

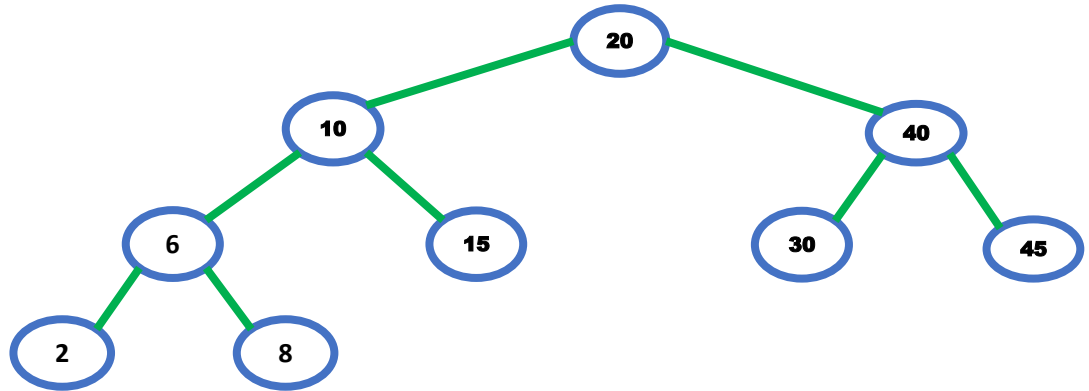
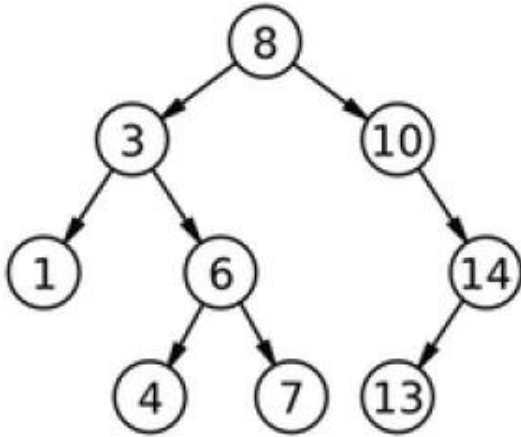


# Binary Search Tree

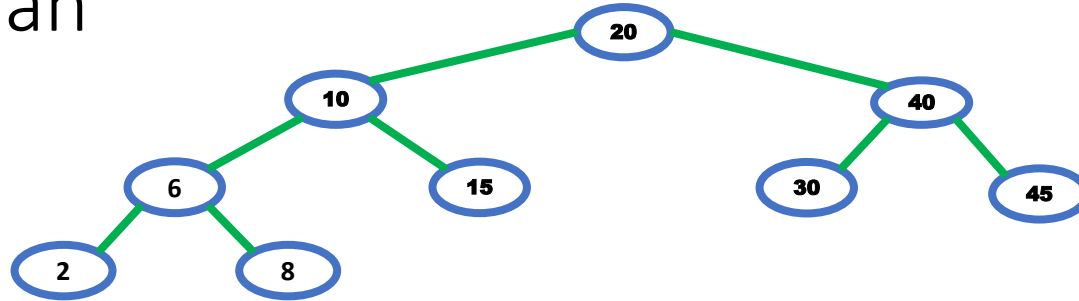
# Binary Search Tree (BST)

- *Binary Search Tree* adalah salah satu bentuk khusus dari *binary tree* dengan sifat bahwa semua *left-child* **harus lebih kecil** daripada *right-child* dan *parent*-nya.
- *Binary search tree* dibuat untuk mengatasi **kelemahan** pada *binary tree* biasa, yaitu kesulitan dalam *searching*/pencarian *node* tertentu dalam *binary tree*.
- Biasa disebut juga dengan *ordered Binary tree* yaitu *binary tree* yang seluruh **children** dari tiap **node** teratur.

# Contoh *Binary Search Tree*



# Pertanyaan



Berdasarkan contoh tree di atas.

- Bagaimana jika ada penambahan node dengan nilai 13?
- Bagaimana jika ada penambahan node lagi dengan nilai 50?
- Bagaimana jika ada penambahan node lagi dengan nilai 27?

**Jawab:** Dengan cara melakukan penelusuran setiap elemen node pada tree sebelum melakukan penambahan dengan cara traversal.

*Binary Tree Traversal*

# Binary Tree Traversal



# Binary Tree Traversal

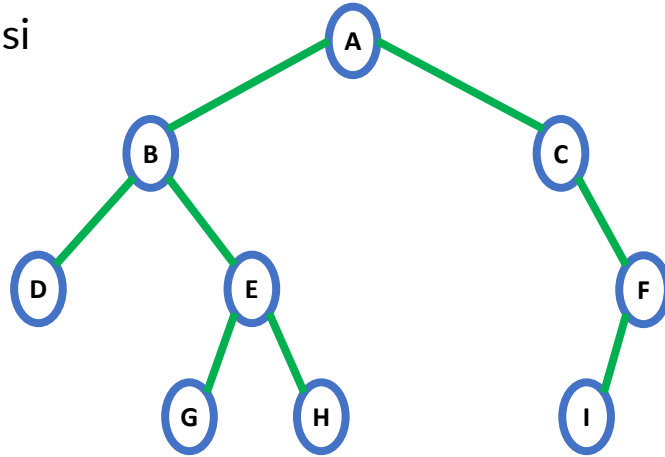
- Merupakan metode penelusuran seluruh node pada binary tree.
- Terdapat beberapa metode, diantaranya:
  - Preoder
  - Inorder
  - Postorder
  - Level order

# Binary Tree Traversal - Preorder

- Langkah-Langkah *Preorder Traversal*:
  1. Kunjungi dan cetak data pada *root*
  2. Secara rekursif kunjungi dan cetak seluruh data pada *subtree* sebelah **kiri** dimulai dari *left-child*.
  3. Secara rekursif kunjungi dan cetak seluruh data pada *subtree* sebelah **kanan** dimulai dari *left-child*.

# Binary Tree Traversal - Preorder (cont.)

- Contoh ilustrasi



Preorder Traversal → A, B, D, E, G, H, C, F, I

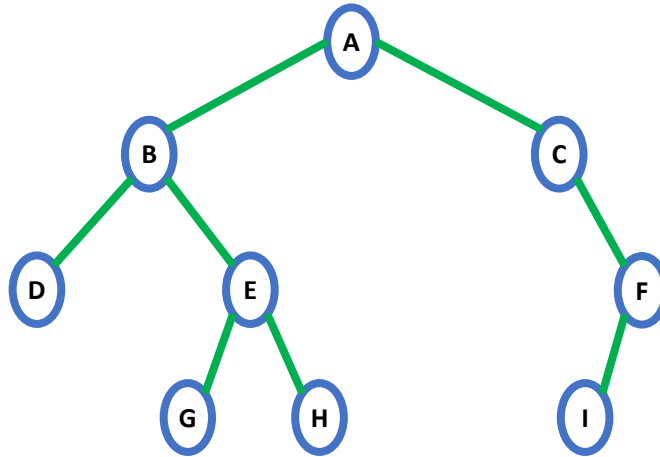
Video Ilustrasi : <https://youtu.be/1WxLM2hwL-U>

# Binary Tree Traversal - Inorder

- Langkah-Langkah *Inorder Traversal*:
  1. Secara rekursif kunjungi dan cetak seluruh data pada *subtree* sebelah **kiri**.
  2. Kunjungi dan cetak data pada *root*
  3. Secara rekursif kunjungi dan cetak seluruh data pada *subtree* sebelah **kanan**.

# Binary Tree Traversal - Inorder (cont.)

- Contoh ilustrasi



Inorder Traversal → D, B, G, E, H, A, C, I, F

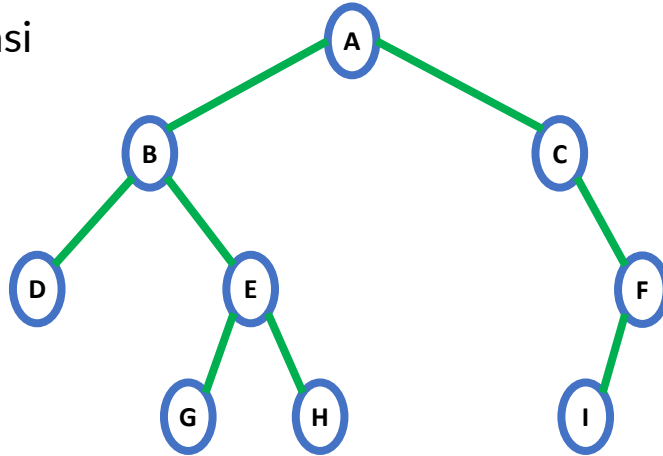
Video Ilustrasi : <https://youtu.be/5dySuyZf9Qg>

# Binary Tree Traversal - Postorder

- Langkah-Langkah *Postorder Traversal*:
  1. Secara rekursif kunjungi dan cetak seluruh data pada *subtree* sebelah **kiri**.
  2. Secara rekursif kunjungi dan cetak seluruh data pada *subtree* sebelah **kanan**.
  3. Kunjungi dan cetak data pada *root*

# Binary Tree Traversal - Postorder (cont.)

- Contoh ilustrasi



Postorder Traversal → D, G, H, E, B, I, F, C, A

Video Ilustrasi : <https://youtu.be/4zVdfkpcT6U>

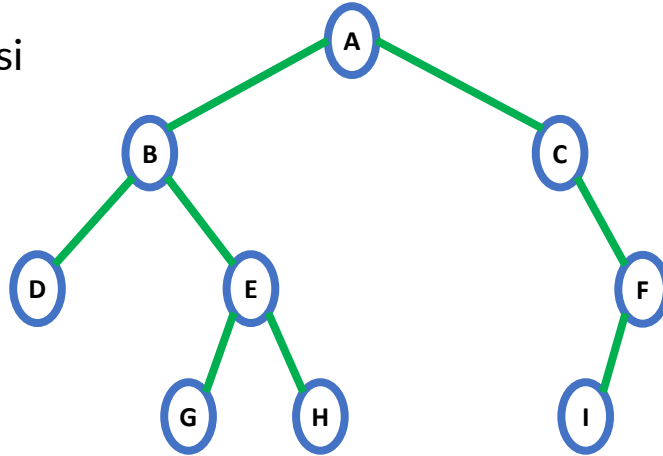
# Binary Tree Traversal – Level order

- Level order juga disebut sebagai Breath First Order
- Langkah-Langkah *Postorder Traversal*:
  1. Kunjungi dan cetak data pada *root*
  2. Secara rekursif kunjungi node-node yang berada di bawah root mulai dari yang paling kiri sampai yang paling kanan
  3. Lakukan Kembali poin ke-2 sampai level yang paling bawah.



# Binary Tree Traversal – Level order (cont.)

- Contoh ilustrasi

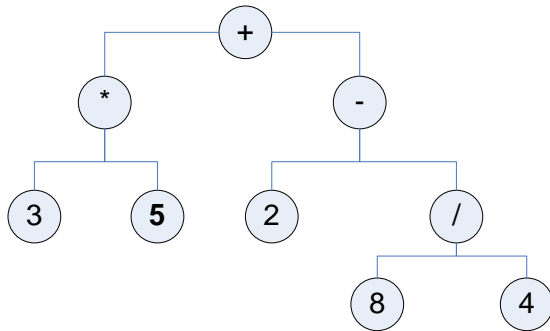


Level order Traversal → A, B, C, D, E, F, G, H, I

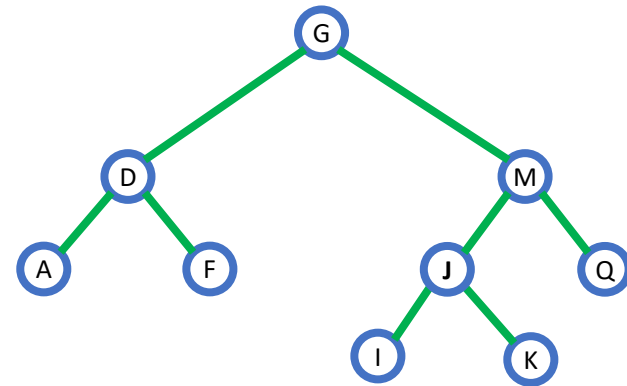
Video Ilustrasi : <https://youtu.be/lozGo2kwRYE>

# Tugas Latihan 3

- Telusuri pohon biner berikut dengan menggunakan metode *preorder*, *inorder*, *postorder*, dan *level order traversal*.



Soal 1



Soal 2

# Operasi Pada Binary Search Tree

# Operasi pada BST

Dalam BST terdapat beberapa operasi yang digunakan untuk mengelola binary tree, diantaranya:

- **Find** : operasi pencarian data pada tree yang sama dengan n (nilai yang dicari).
- **Insert** : Operasi penambahan data/node pada tree
- **Delete** : Operasi penghapusan data/node pada tree
- **Display** : Operasi menampilkan data

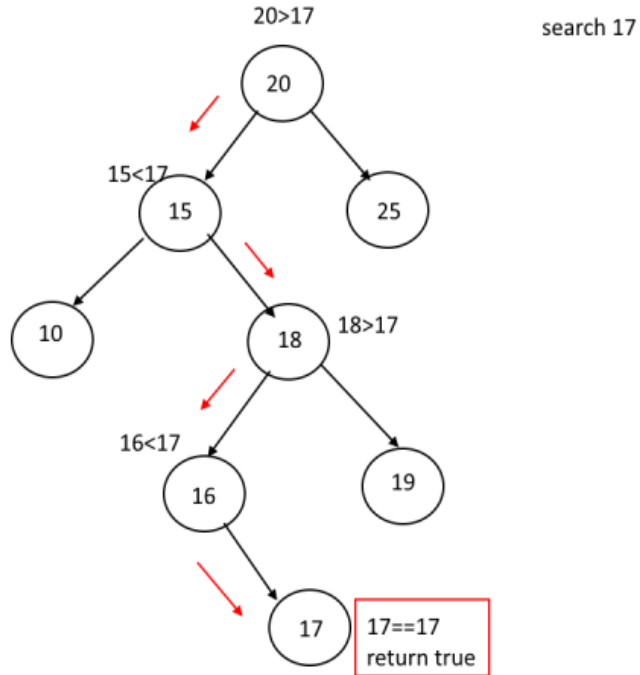


# Operasi - *Find*

1. Mulai dari *root*, dan bandingkan nilai *root* dengan *n* (nilai yang dicari).
2. Jika nilai *n* lebih kecil dari *root*, lakukan pencarian pada *subtree* sebelah kiri dari *root*.
3. Jika nilai *n* lebih besar dari *root*, lakukan pencarian pada *subtree* sebelah kanan dari *root*.
4. Lakukan pencarian pada *node-node* di bawah *root* dengan cara yang sama seperti langkah nomor 2 & 3.
5. Jika ketemu, *return true*.
6. Jika pada akhir *node (leaf)* tidak ketemu, *return false*.

# Operasi – *Find* (cont.)

## ► Ilustrasi



Video ilustrasi *Find*/search element : <https://youtu.be/a7xOXL7hn94>

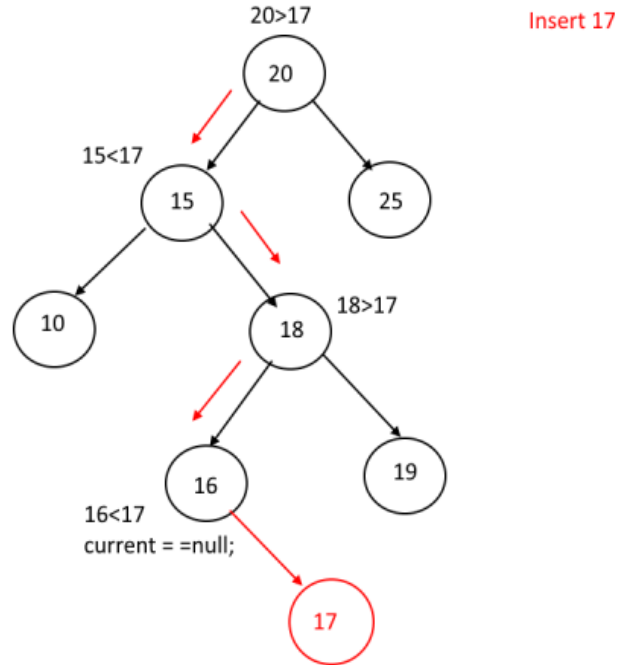
# Operasi – *Insert*

Operasi insert mirip dengan operasi *find*. Untuk melakukan *insert* sebuah *node*, pertama kita harus mencari tempat yang tepat untuk meletakkan node tersebut. Langkah-langkahnya:

1. Buat variabel *current* bertipe *node*, set *current* dengan *root*.
2. Bandingkan nilai *current* dengan *n* (nilai yang akan dimasukkan).
3. Jika nilai *current* lebih besar dari *n*, lakukan pencarian tempat pada sebelah kiri dari *root*.
4. Jika nilai *current* lebih kecil dari *n*, lakukan pencarian tempat pada sebelah kanan dari *root*.
5. Jika ketemu nilai *current* *null*, yang artinya pencarian sudah sampai pada *leaf* (akhir *node*), letakkan node yang baru berisikan *n* tadi ke posisi *current* tersebut.

# Operasi – *Insert* (cont.)

## ► Ilustrasi



Video ilustrasi *Insert* element : <https://youtu.be/WoRa9vnHkDM>



# Operasi – *Delete*

Kemungkinan skenario delete yang bisa dilakukan:

1. Node yang akan dihapus adalah *node leaf* (*no children*).
2. Node yang akan dihapus hanya memiliki satu child.
3. Node yang akan dihapus memiliki dua child.

Video ilustrasi *Delete element* : <https://youtu.be/c1zKNiABiHk>

# Operasi – *Delete* (skenario 1)

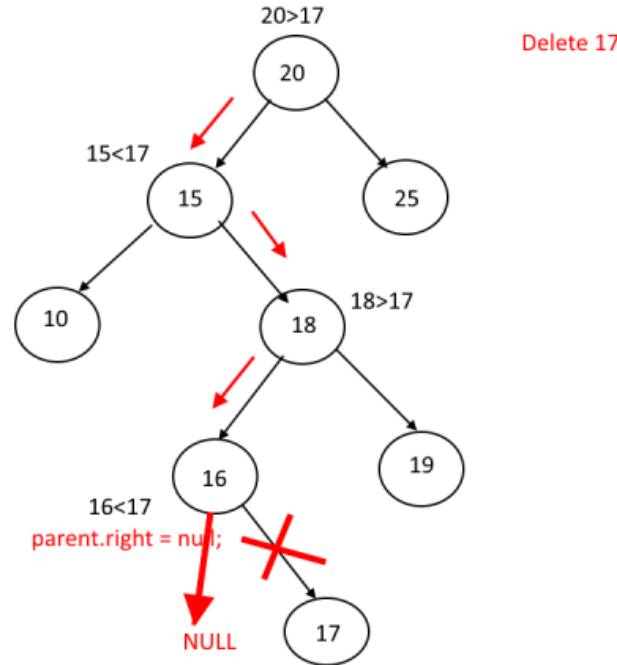
**Node yang akan dihapus adalah *node leaf* (no children).**

Langkah-langkah:

1. Lakukan traverse pada tree dengan cara membandingkan tiap node yang dikunjungi.
2. Jika node yang dikunjungi lebih besar dari  $n$  (nilai node yang akan dihapus), lanjutkan traverse ke sebelah kiri.
3. Jika node yang dikunjungi lebih kecil dari  $n$ , lanjutkan traverse ke sebelah kanan.
4. Jika ketemu, set pointer
  - $left = null \rightarrow$  jika nilai  $n$  (node yang akan dihapus atau *child*) lebih kecil daripada *parent*, atau
  - $right = null \rightarrow$  jika nilai  $n$  (node yang akan dihapus atau *child*) lebih besar daripada *parent*.

# Operasi – *Delete* (skenario 1) (cont.)

## Ilustrasi



Case 1 : Node to be deleted is a leaf node ( No Children).

# Operasi – *Delete* (skenario 2)

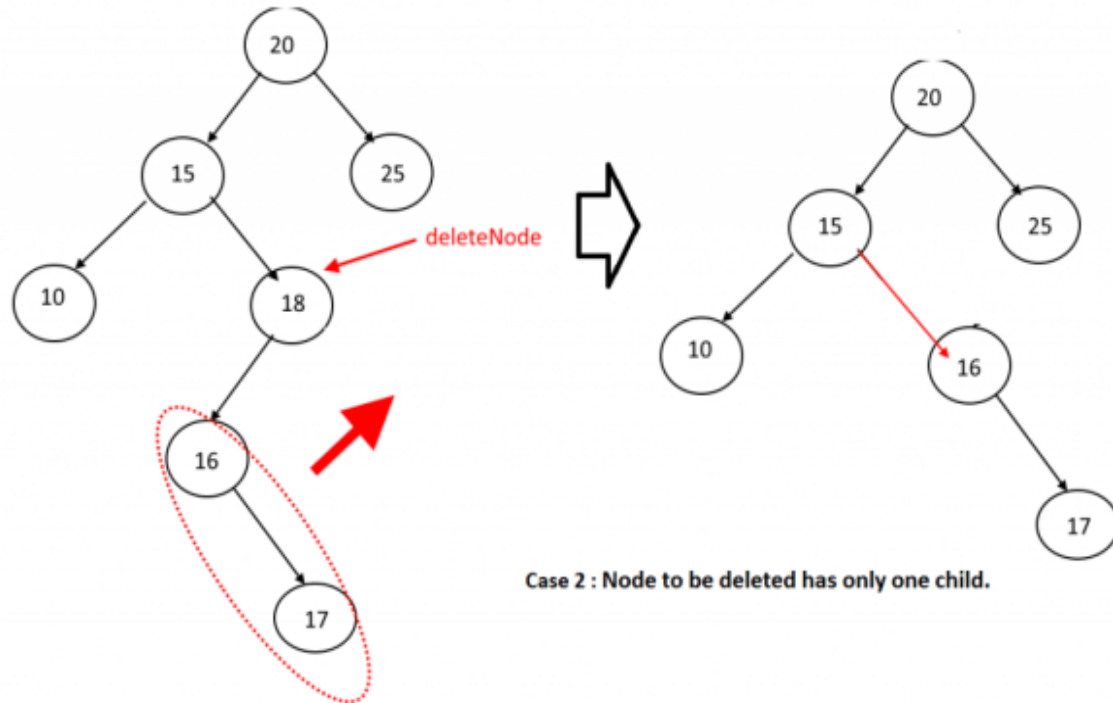
**Node yang akan dihapus memiliki satu child.**

Langkah-langah:

1. Lakukan traverse menuju node yang akan dihapus.
2. Jika ketemu, catat parent dari node tersebut, dan ada disebelah mana node tersebut dari parent-nya (kanan atau kiri).
3. Dari node tersebut, cek sebelah mananya yang null (karena node tersebut hanya memiliki satu child).
4. Misal node yang akan dihapus tersebut memiliki child di sebelah kiri. Maka set child dari node tersebut (beserta sub tree nya) dan letakkan di sisi parent menggantikan posisi node yang akan dihapus tadi.

# Operasi – *Delete* (skenario 2) (cont.)

## Ilustrasi



# Operasi – *Delete* (skenario 3)

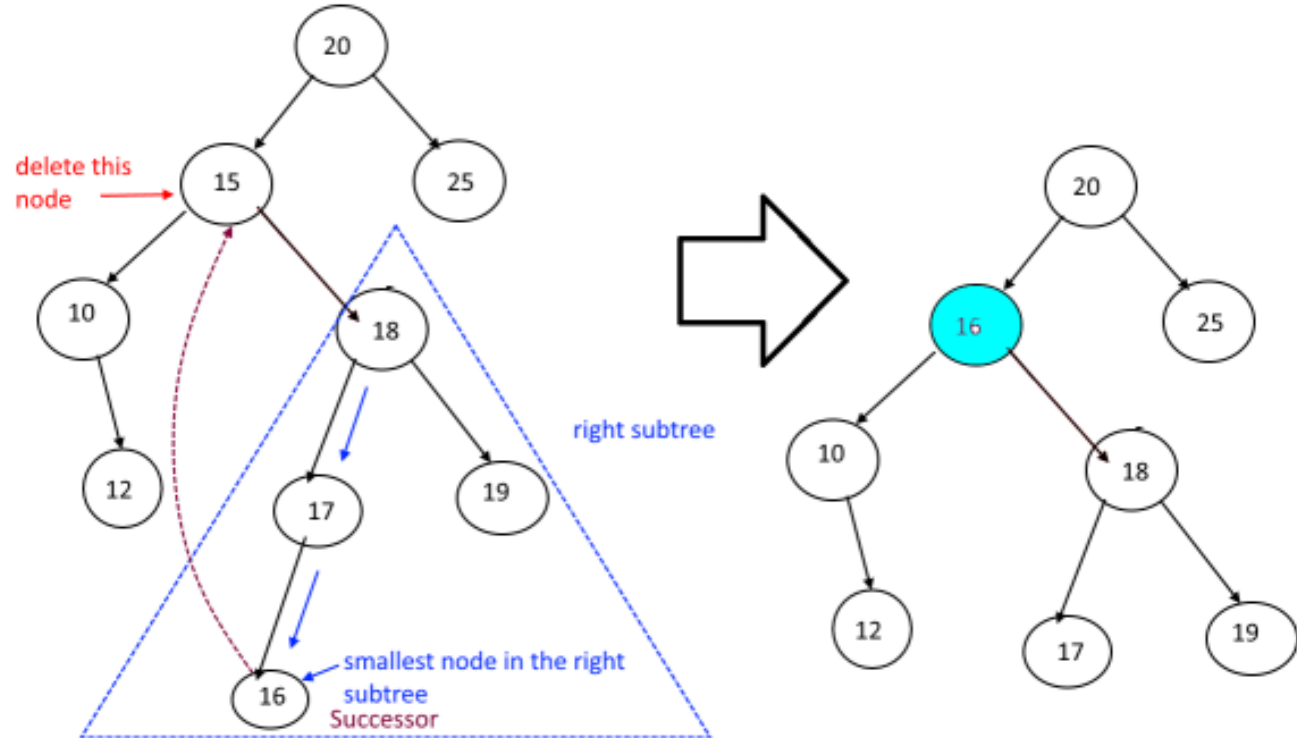
**Node yang akan dihapus memiliki dua child.**

Langkah-langah:

1. Cari successor dari node yang akan dihapus, untuk menggantikan node yang akan dihapus tersebut.
2. Successor adalah node paling kecil dari subtree sebelah kanan pada node yang akan dihapus.

# Operasi – *Delete* (skenario 3) (cont.)

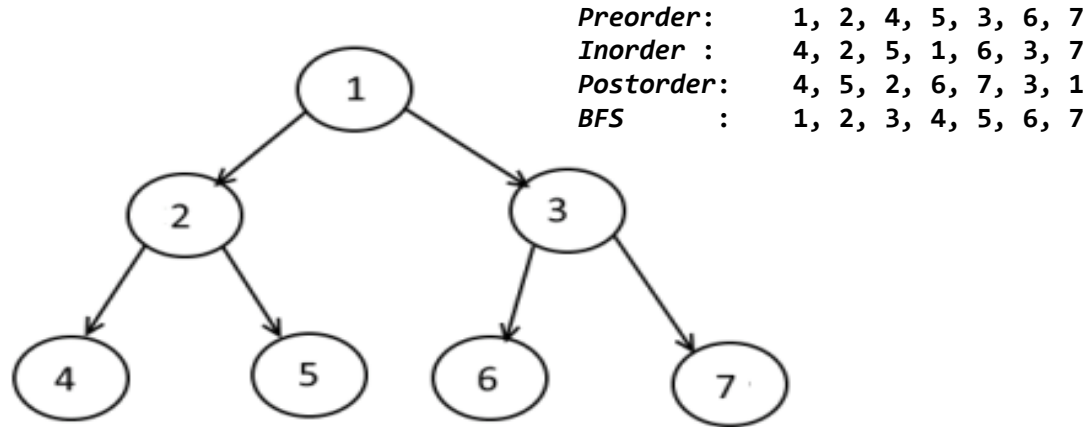
## Ilustrasi



# Operasi – *Display*

Mencetak semua node-node yang ada pada tree. Proses mencetak data menggunakan metode traversal yang sudah dipelajari, diantaranya metode

- *Preorder*
- *Inorder*
- *Postorder*
- *Level order / BFS*

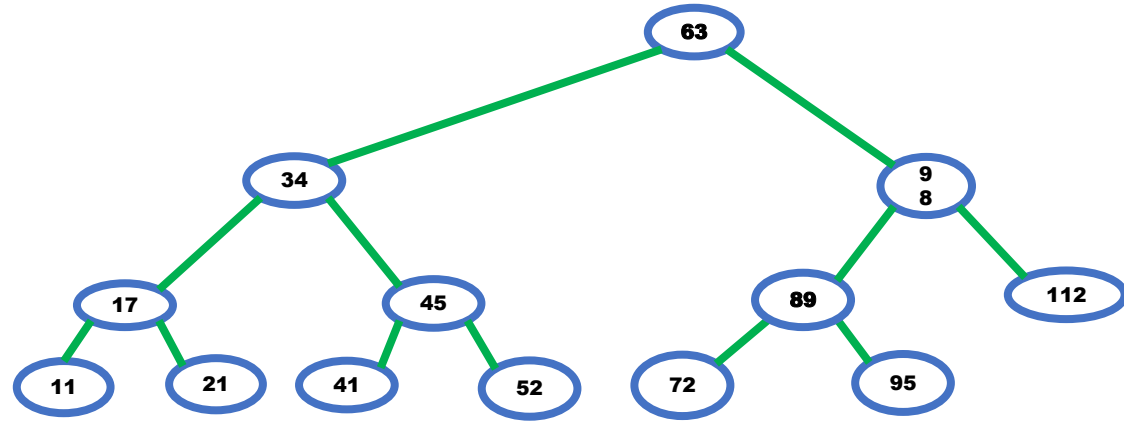


Untuk video ilustrasi *Display element* mengikuti video ilustrasi *Binary Tree Traverse*



# Tugas Latihan 4

Terdapat sebuah tree seperti gambar disamping.



Terdapat data baru (**40**) yang akan ditambahkan dan data lama (**98**) yang akan dihapus.

Ilustrasikan operasi (find, insert, delete, display) yang akan dilakukan untuk mengatasi penambahan dan penghapusan data tersebut.

# Tugas Latihan

Silahkan kerjakan Tugas Latihan yang ada pada materi ini

1. Tugas Latihan 1 (Hal. 26)
2. Tugas Latihan 2 (Hal. 35)
3. Tugas Latihan 3 (Hal. 50)
4. Tugas Latihan 4 (Hal. 65)

