

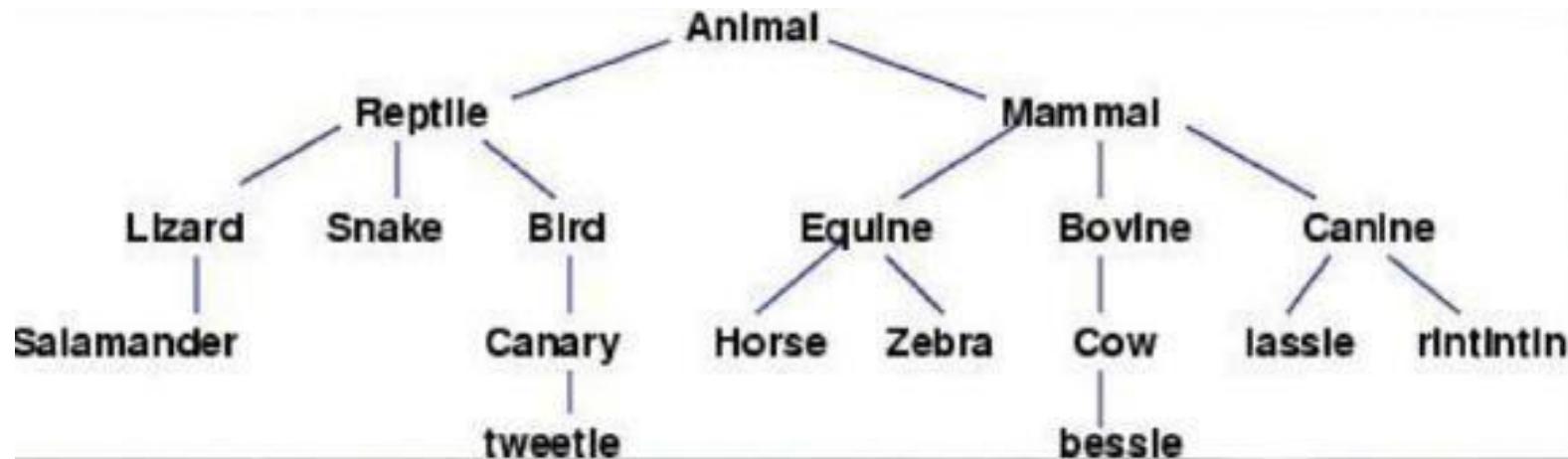
PERTEMUAN 12

Binary Tree & Binary Search Tree (BST)

Hermanto, S.Kom. M.Kom

Defenisi Tree (Pohon)

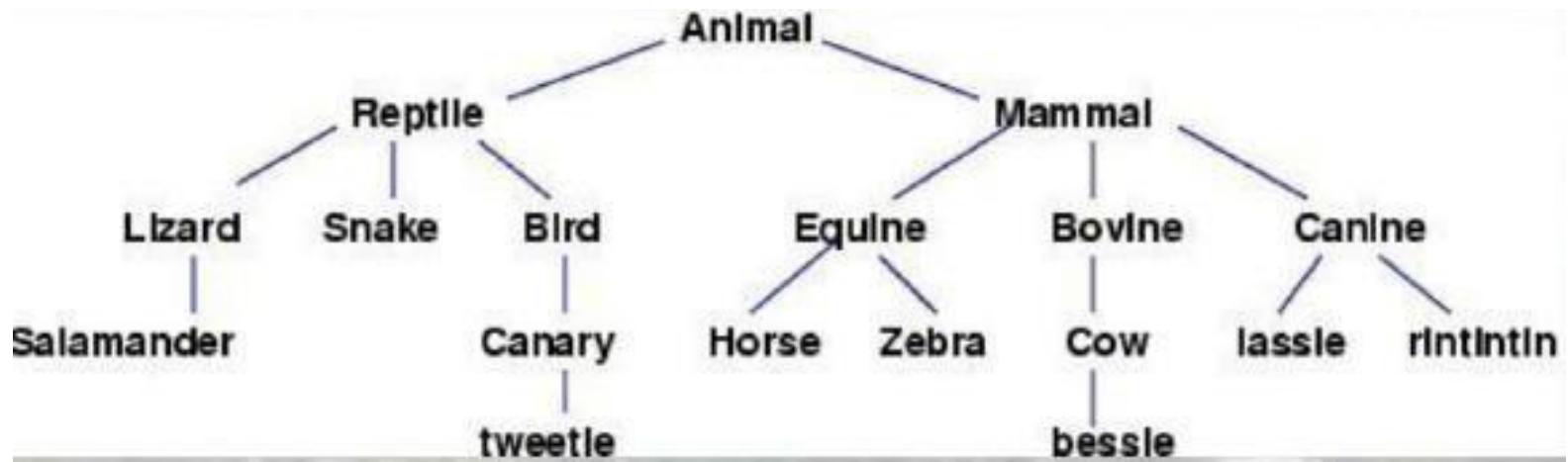
- Tree adalah Kumpulan elemen yang saling terhubung secara hirarki (one to many).
- Elemen pada tree disebut node.



Aturan

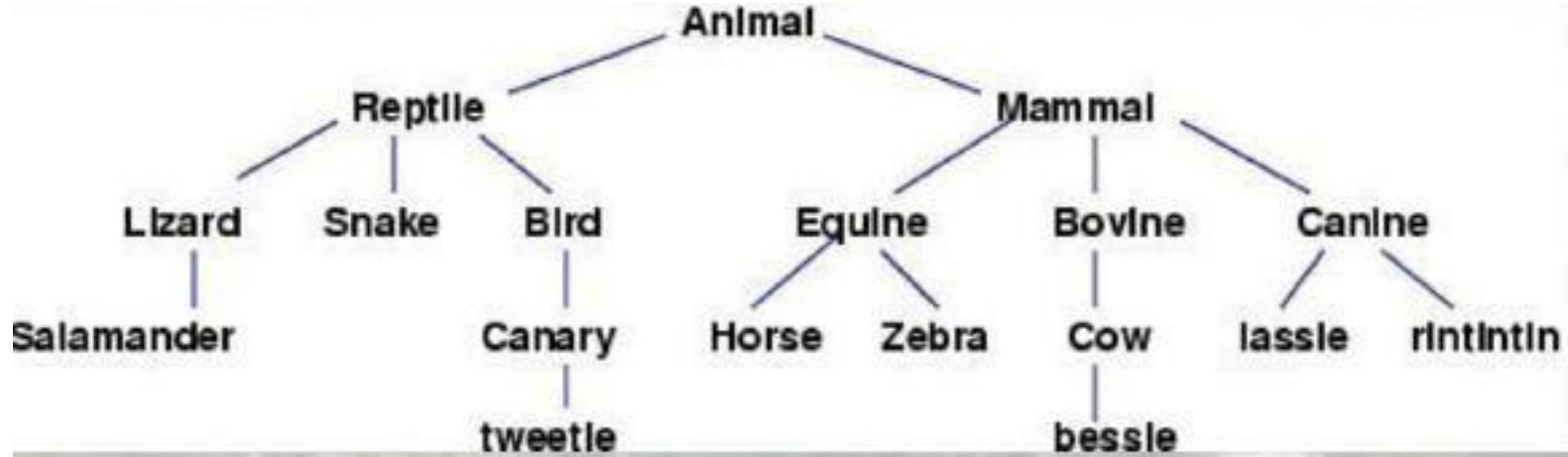
- ❑ Sebuah node hanya boleh memiliki satu induk/parent.
Kecuali root, tidak memiliki induk/parent.
- ❑ Setiap node dapat memiliki nol atau banyak cabang anak
(one to many).
- ❑ Node yang tidak memiliki cabang anak disebut **daun**.

Contoh Tree



- Ada berapa node pada tree diatas?
- Node manakah yang menjadi root?
- Ada berapa jumlah node leaf/daun pada tree tersebut?

Contoh Tree

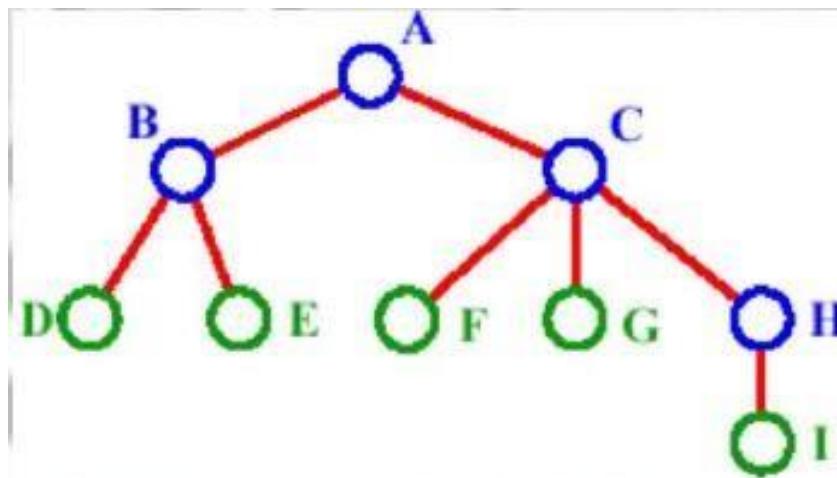


- Ada berapa node pada tree diatas? **18**
- Node manakah yang menjadi root? **Animal**
- Ada berapa jumlah node leaf/daun pada tree tersebut? **8**

Istilah pada Tree

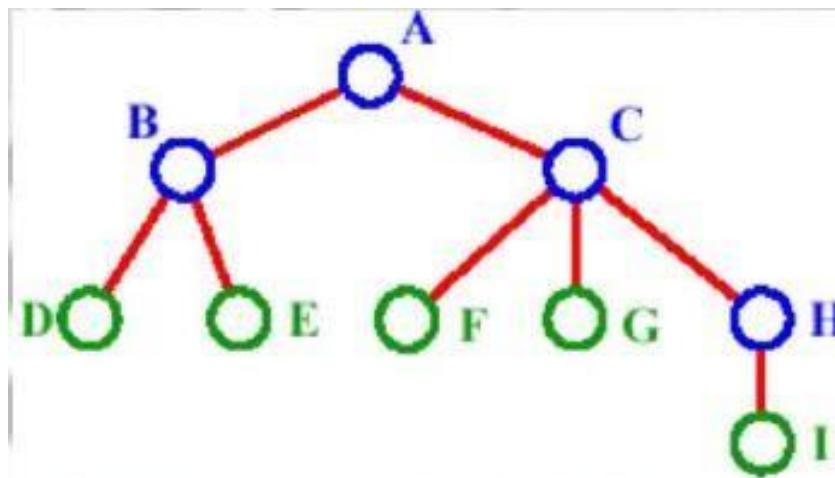
Predecesor	Node yang berada diatas node tertentu.
Successor	Node yang berada dibawah node tertentu.
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendant	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node.
Child	Successor satu level di bawah suatu node.
Sibling	Node-node yang memiliki parent yang sama
Subtree	Suatu node beserta descendantnya.
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan dalam suatu tree
Root	Node khusus yang tidak memiliki predecessor.
Leaf	Node-node dalam tree yang tidak memiliki successor.
Degree	Banyaknya child dalam suatu node

Contoh Latihan



- Predecesor (F)?
- Succesor (B)?
- Ancestor (F)?
- Descendant (B)?
- Parent (I)?
- Child (C)?
- Sibling (G)?
- Size?
- Height?
- Root?
- Leaf?
- Degree (C)?

Jawaban



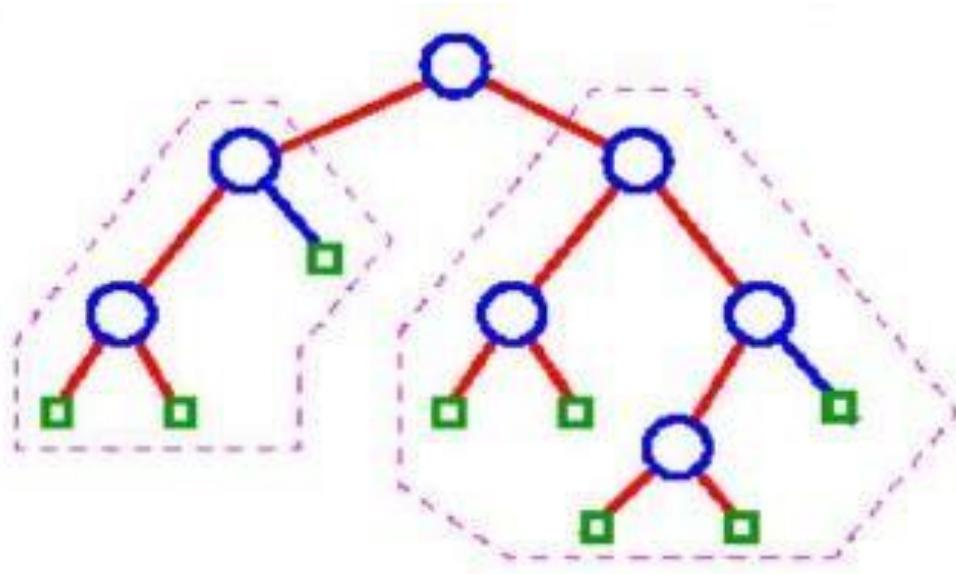
- Predecesor (F)? = A, B, C
- Succesor (B)? = D, E, F, G, H, I
- Ancestor (F)? = C, A
- Descendant (B)? = D, E
- Parent (I)? = H
- Child (C)? = F, G, H
- Sibling (G)? = F, H
- Size? = 9
- Height? = 4
- Root? = A
- Leaf? = D, E, F, G, I
- Degree (C)? = 3

Latihan Pemahaman

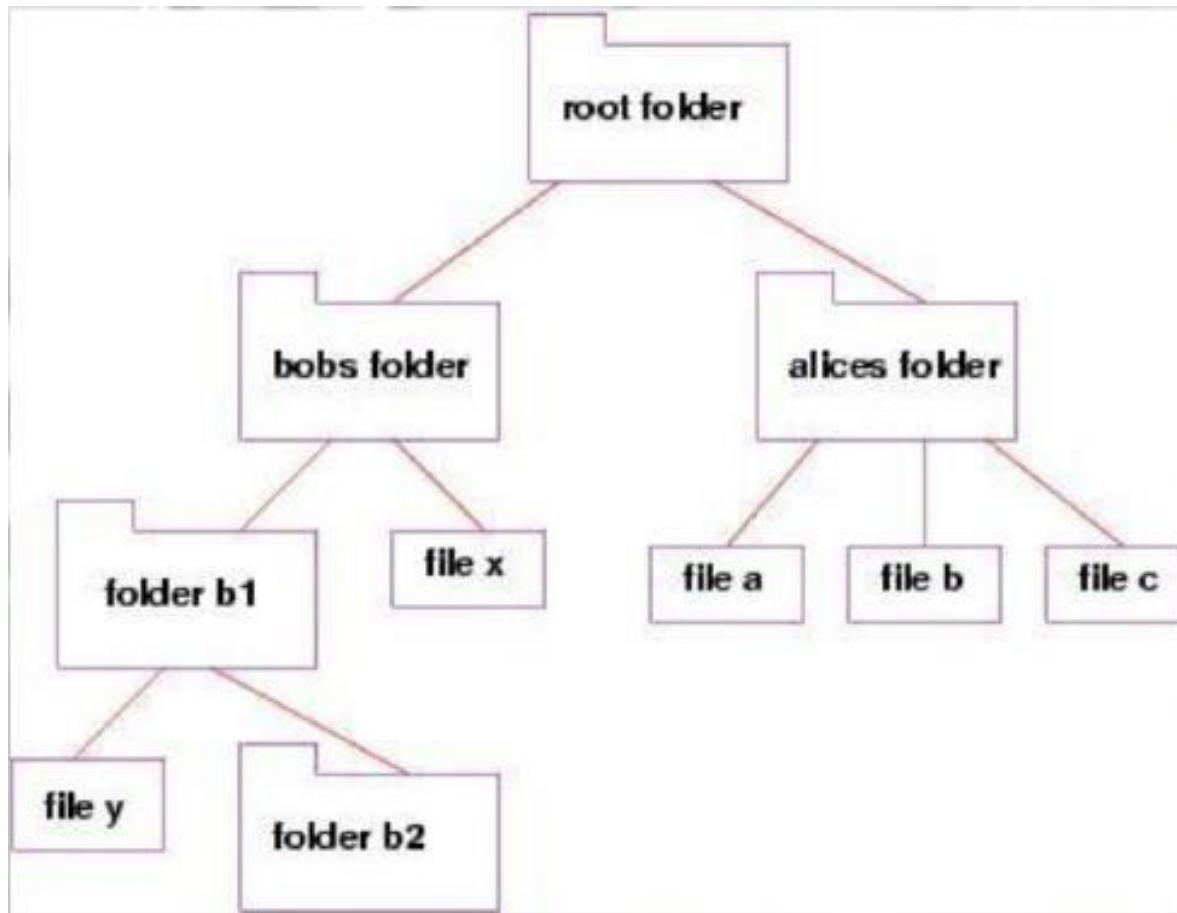
1. Apa itu leaf/daun?
2. Apa itu root?
3. Apa itu level pada tree?
4. Apa itu subtree?
5. Apa itu binary tree?

Binary Tree

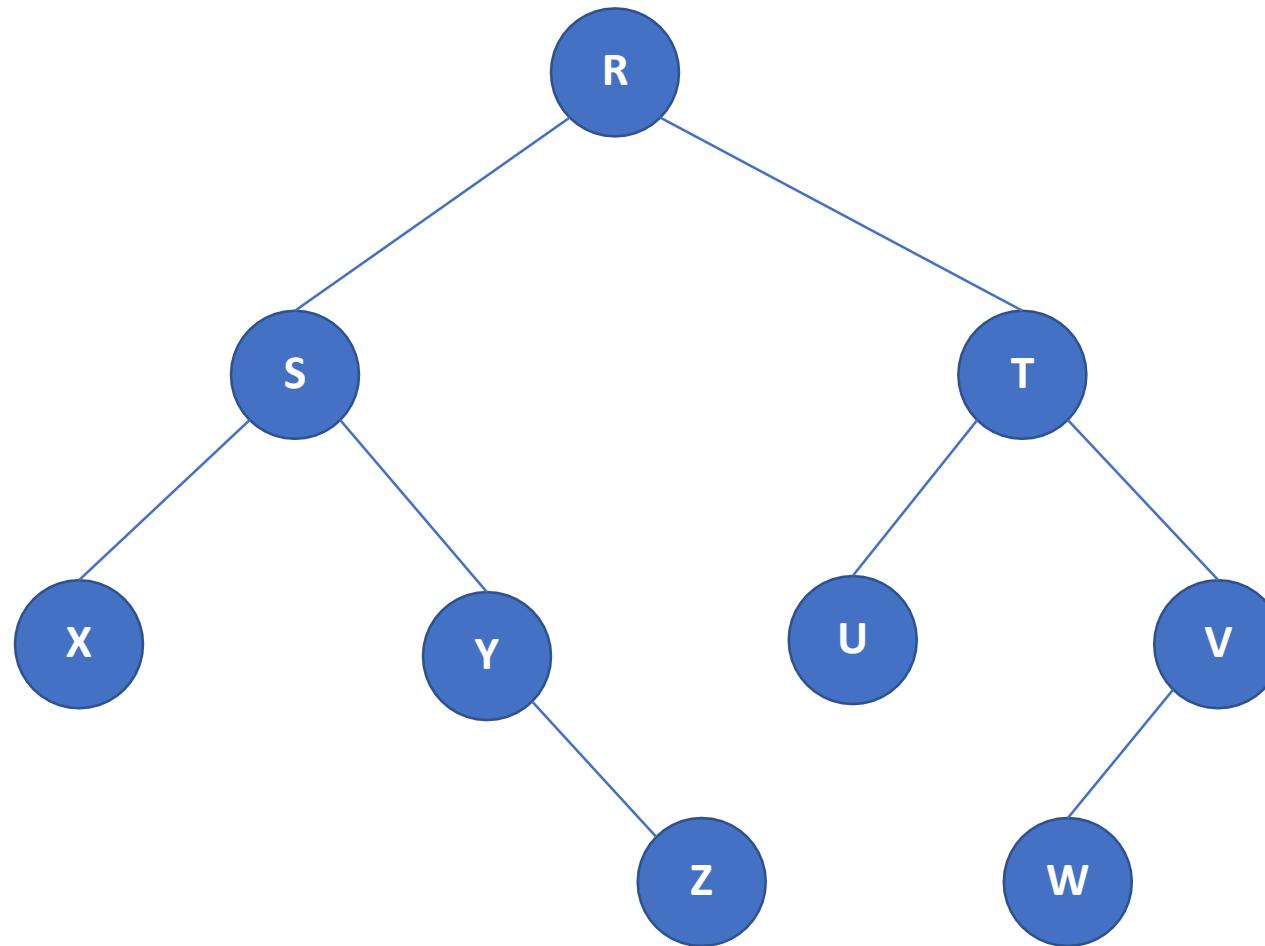
- Binary tree = pohon biner
- Tiap node-nya memiliki maksimal 2 cabang.



Contoh Tree

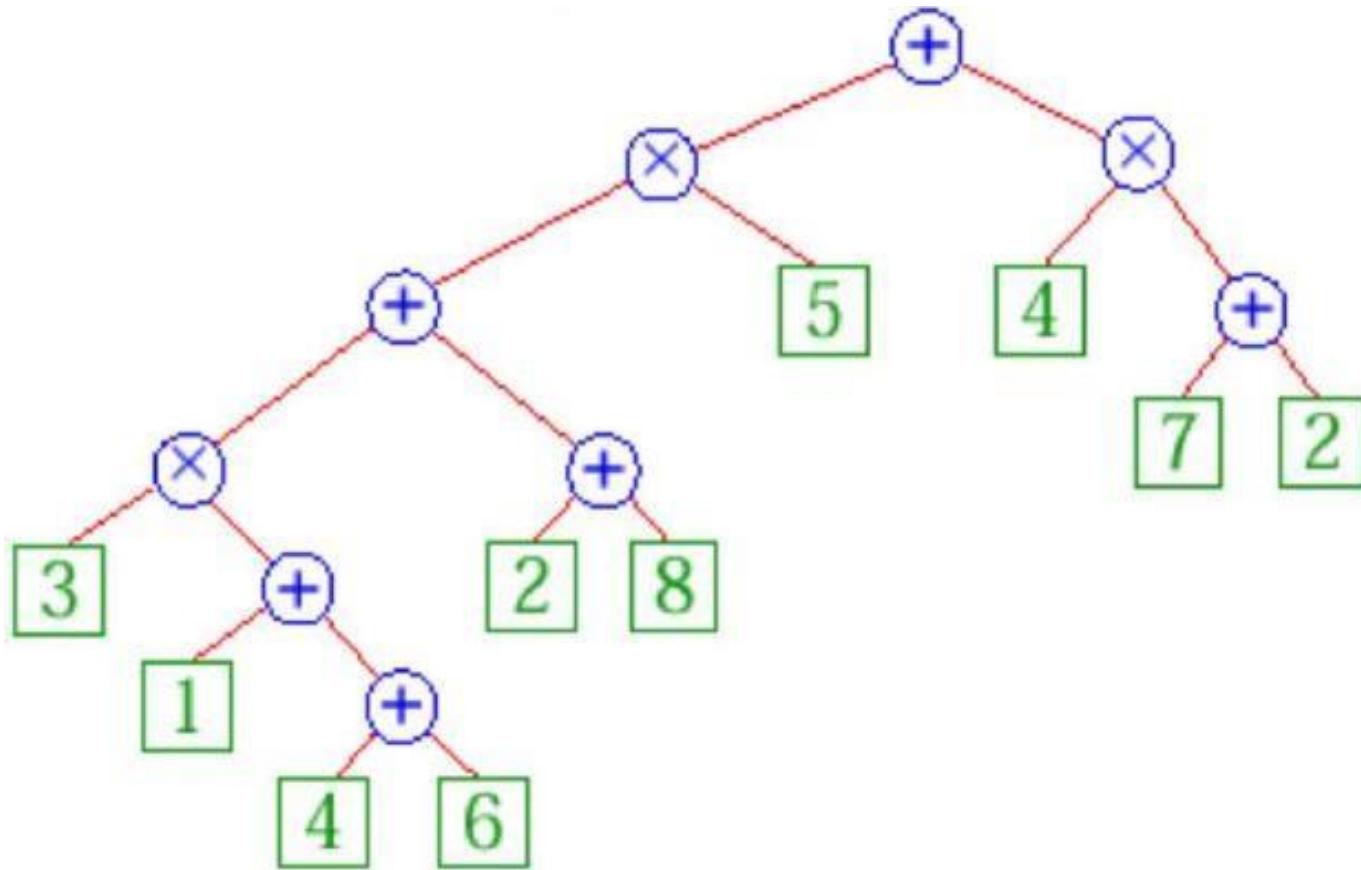


Contoh Binary Tree



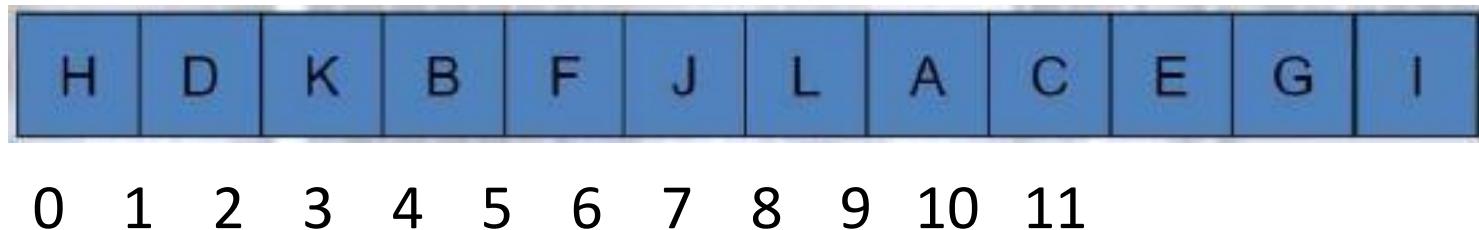
Contoh Binary Tree

- Representasi ekspresi arithmatik

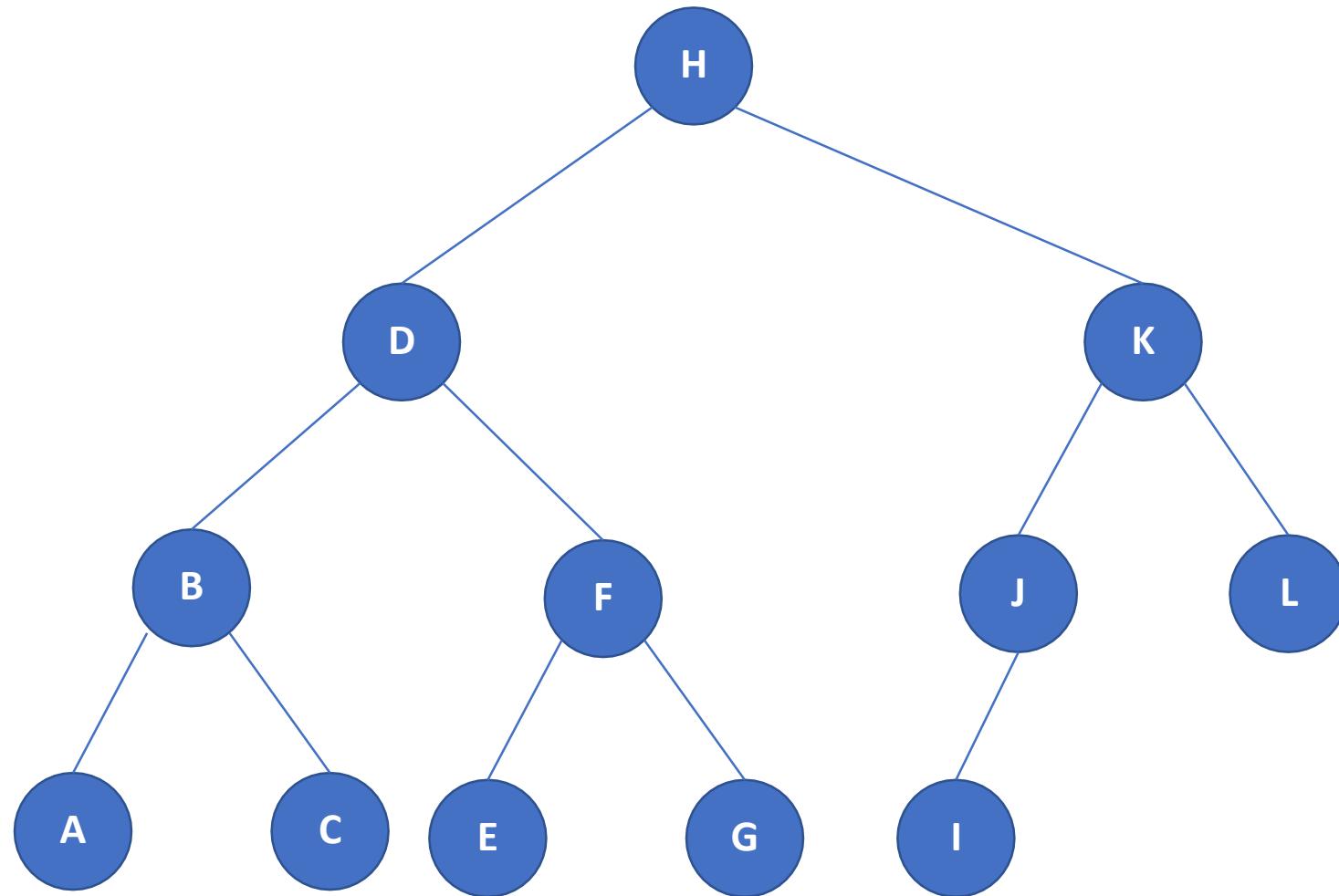

$$((((3 \times (1 + (4 + 6)))) + (2 + 8)) \times 5) + (4 \times (7 + 2)))$$

Representasi Binary Tree

- Binary tree dapat direpresentasikan dengan menggunakan array maupun linked list.
- Representasi binary tree menggunakan array (root pada index 0) :



Contoh



Representasi Binary Tree

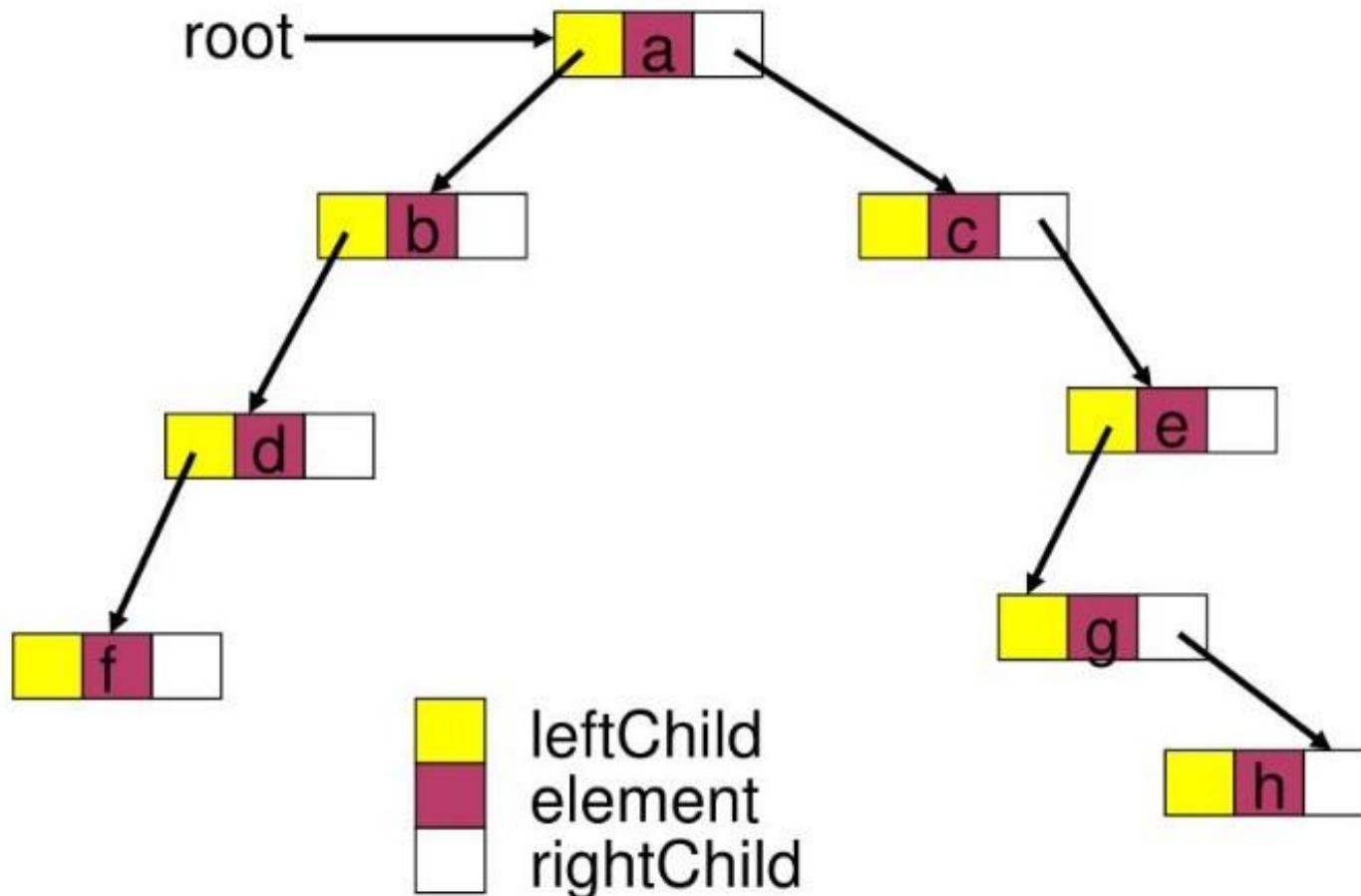
Representasi binary tree menggunakan array (asumsi root pada index 0) :



Representasi binary tree menggunakan array (asumsi root pada index 1) :

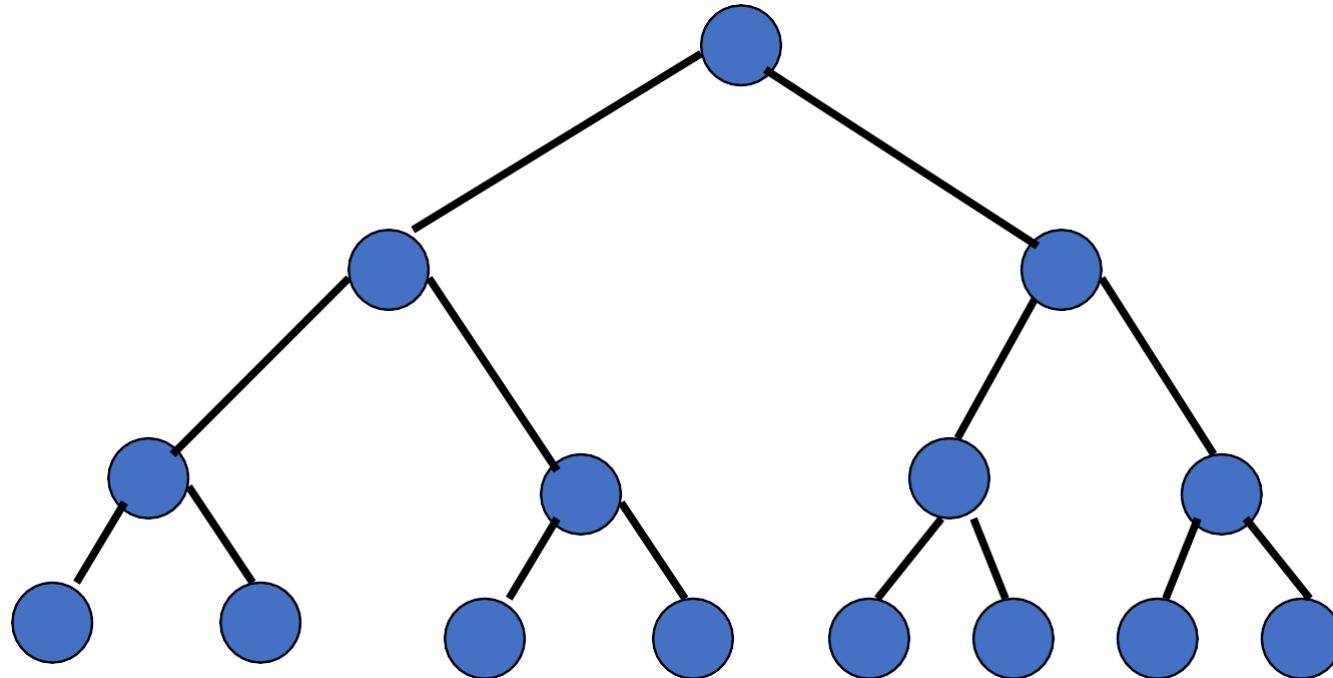


Representasi Linked List



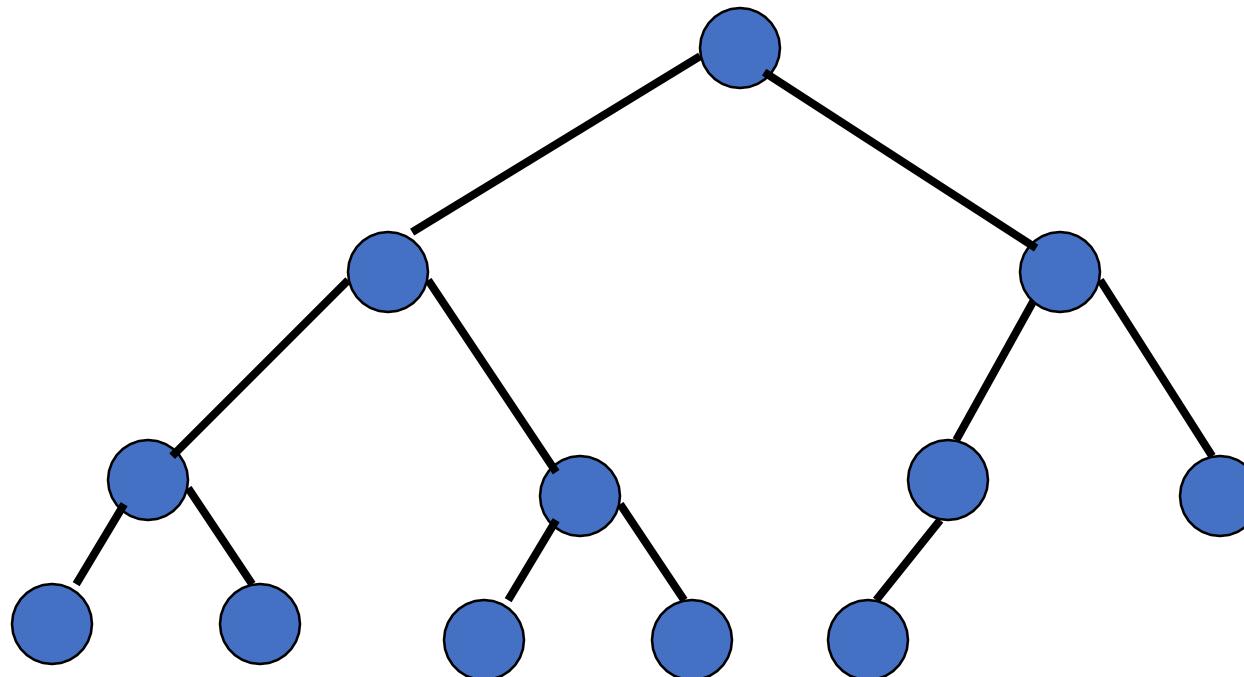
Full Binary Tree

- ❑ Tiap sub tree memiliki panjang path yang sama.
- ❑ Disebut juga maximum binary tree



Complete Binary Tree

- Seluruh node sebelah kiri terisi seluruhnya. Node sebelah kanan pada level $n-1$ ada yang kosong.



Akses Elemen

Posisi node dapat ditentukan berdasarkan rumus berikut :

- ❑ Asumsi root dimulai dari index 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 : $2*i+1$

Akses Elemen

Hubungan Level dengan Jumlah Data :

Level Ke :	Jumlah data	
	Minimum	Maksimum
1	1	$1 = 2^0$
2	1	$2 = 2^1$
3	1	$4 = 2^2$
4	1	$8 = 2^3$
5	1	$16 = 2^4$
6	1	$32 = 2^5$
...
L	1	$2^{(L-1)}$

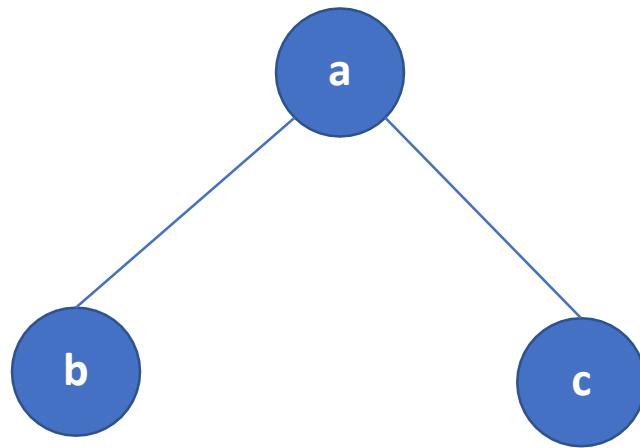
Binary Tree Traversal

- Teknik penelusuran seluruh node pada binary tree.
- Ada 4 metode :
 - PRE-ORDER
 - IN-ORDER
 - POST-ORDER
 - LEVEL-ORDER

Pre-Order Traversal

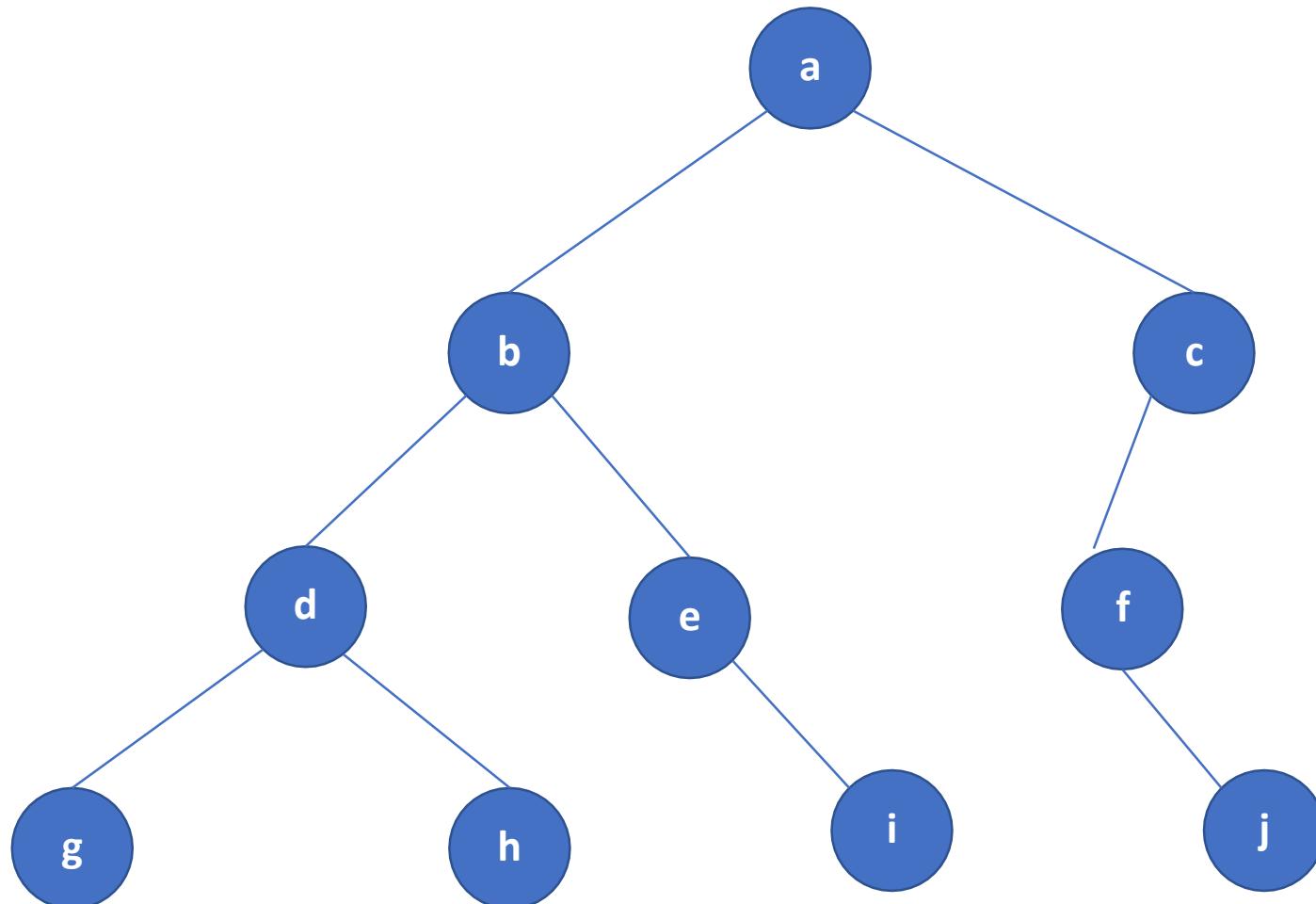
- Algoritma Pre-Order traversal :
 1. Cetak data pada **root**
 2. Secara rekursif mencetak seluruh data pada sub pohon **kiri**
 3. Secara rekursif mencetak seluruh data pada sub pohon **kanan**

Contoh Pre-Order (Visit = print)

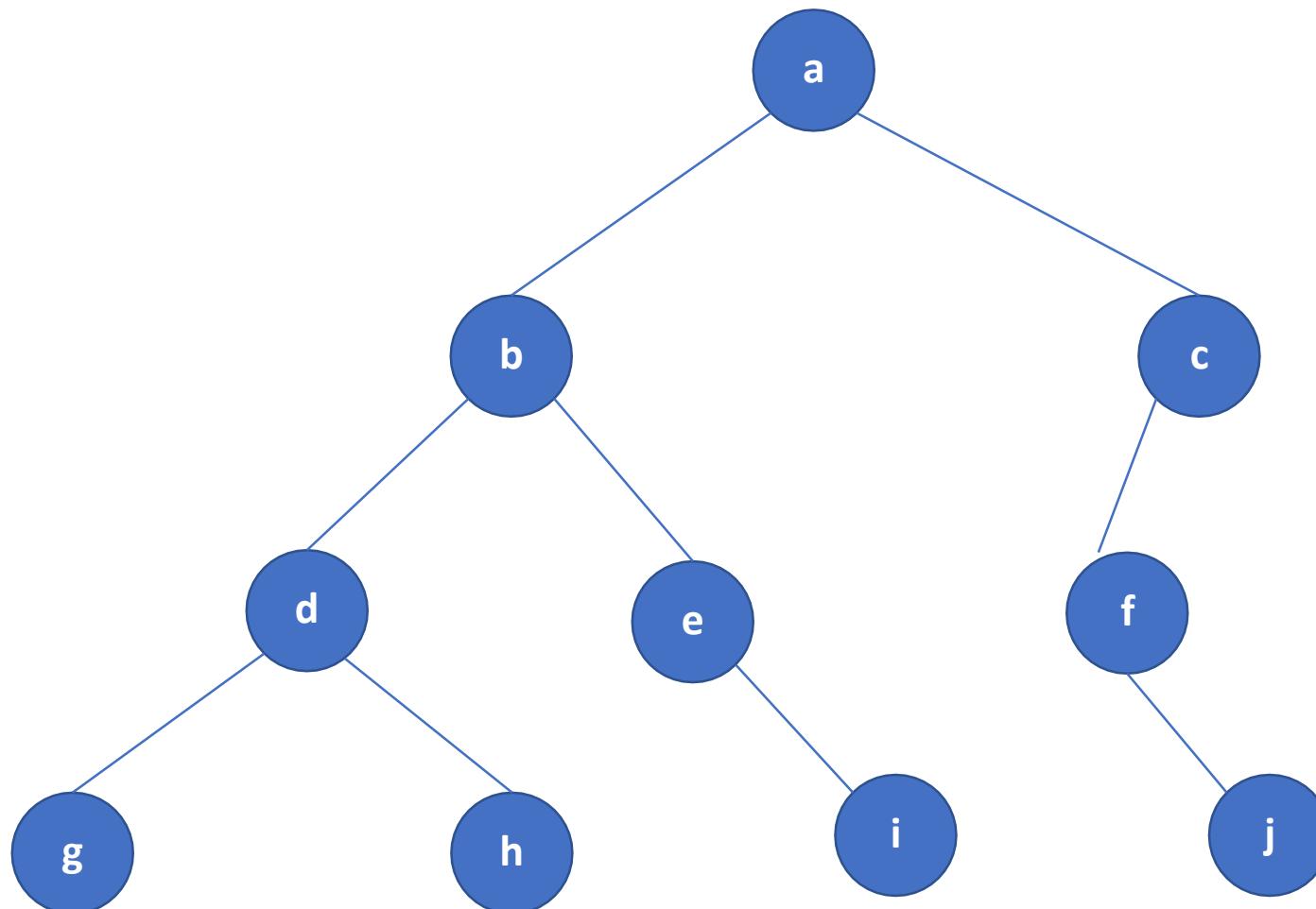


a b c

Contoh Pre-Order (Visit = print)

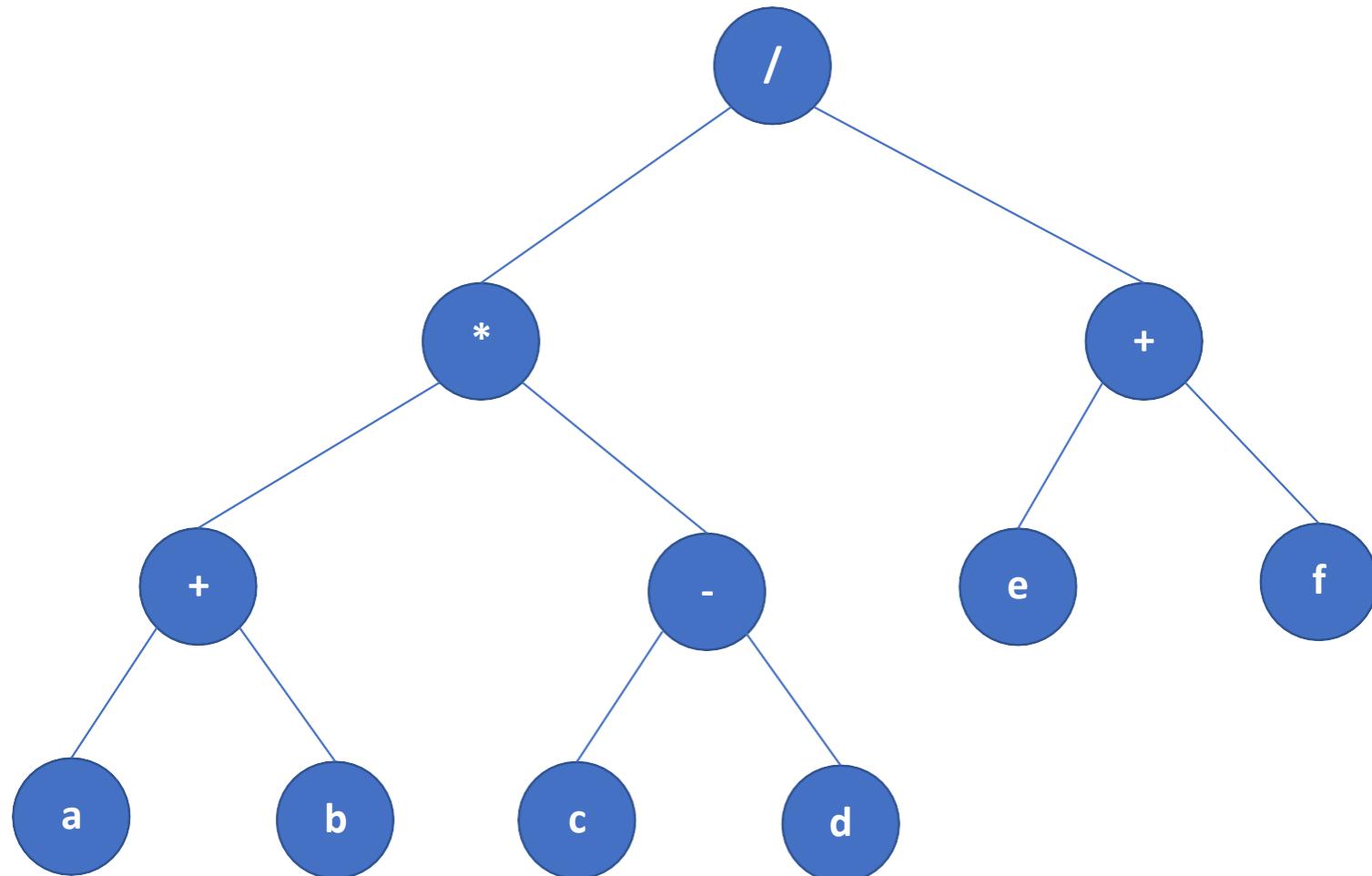


Contoh Pre-Order (Visit = print)



a b d g h e i c f j

Pre-Order Of Expression Tree



/ * + a b - c d + e f

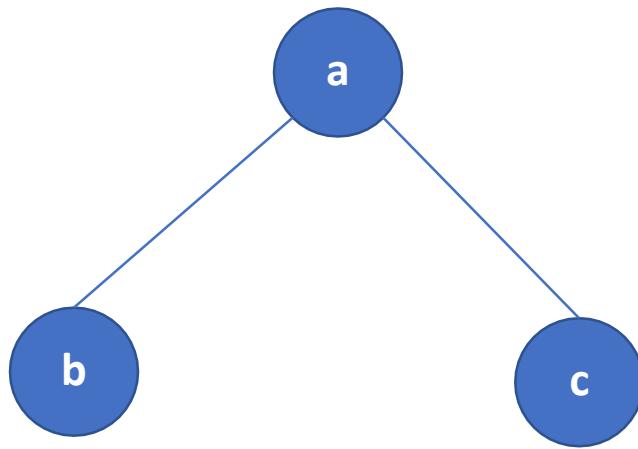
Gives prefix from of Expression!

In-Order Traversal

□ Algoritma In-Order traversal :

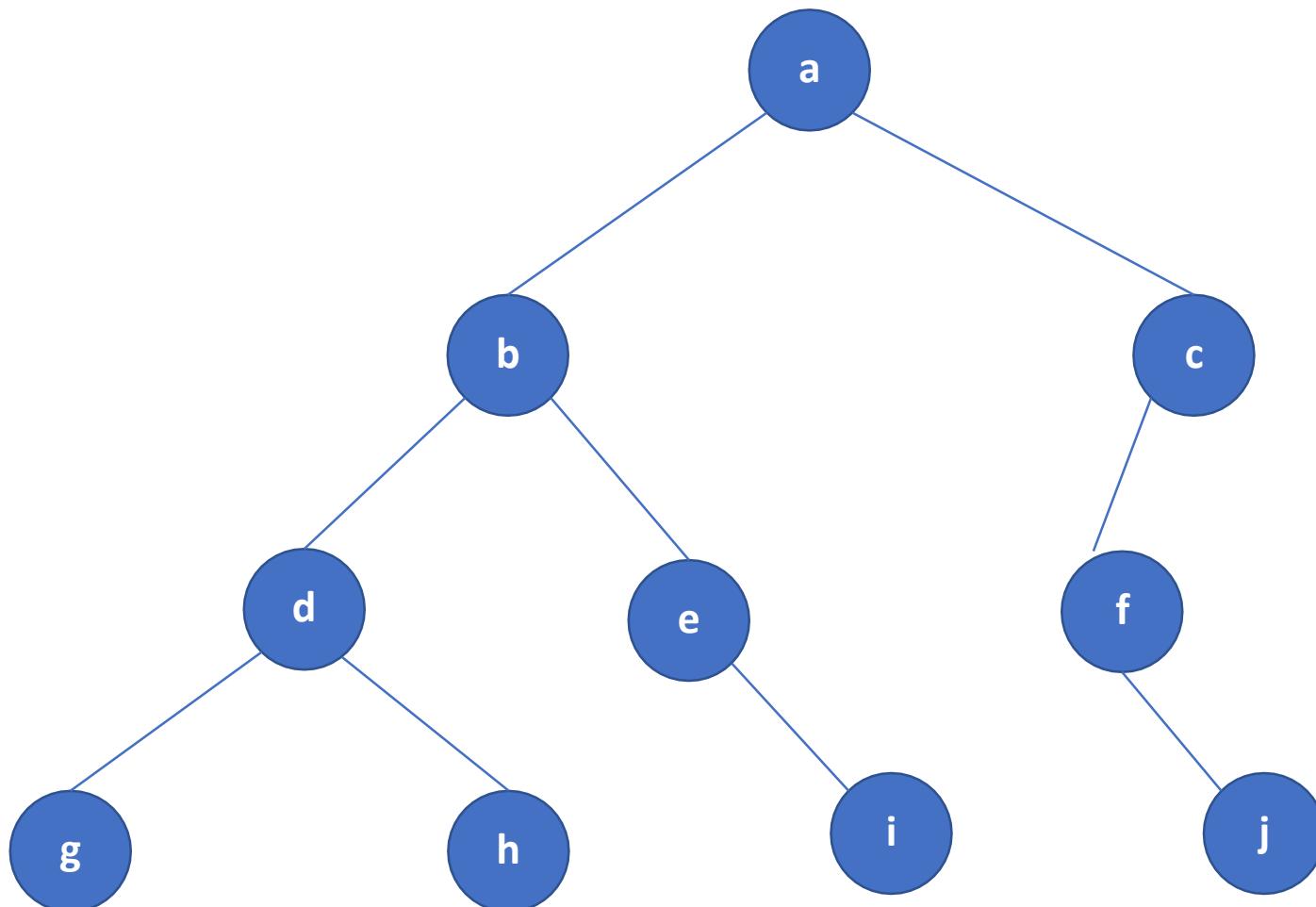
1. Secara rekursif mencetak seluruh data pada sub pohon
 ▶ **kiri**
2. Cetak data pada **root**
3. Secara rekursif mencetak seluruh data pada sub pohon
 kanan

Contoh In-Order (Visit = print)

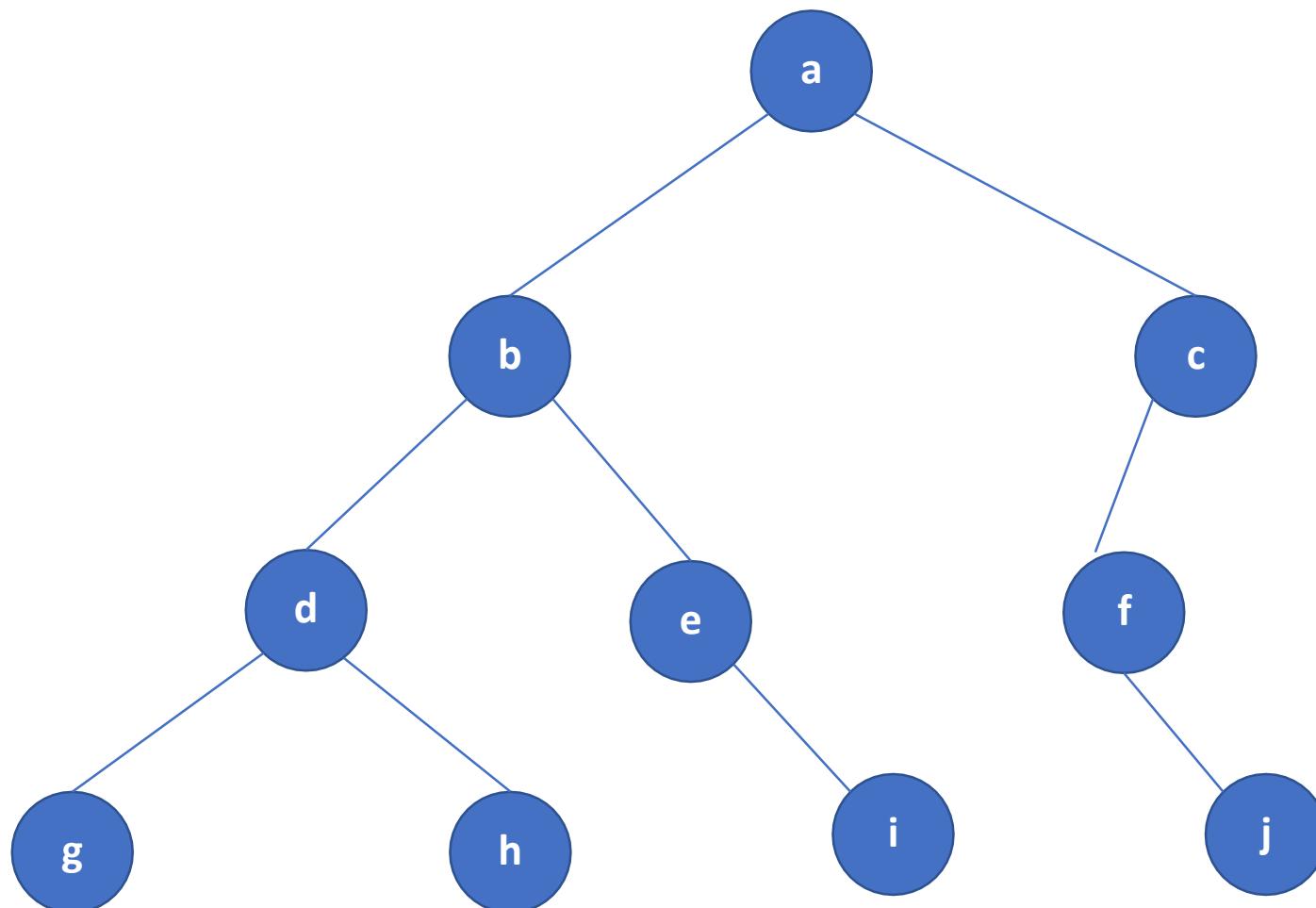


b a c

Contoh In-Order (Visit = print)

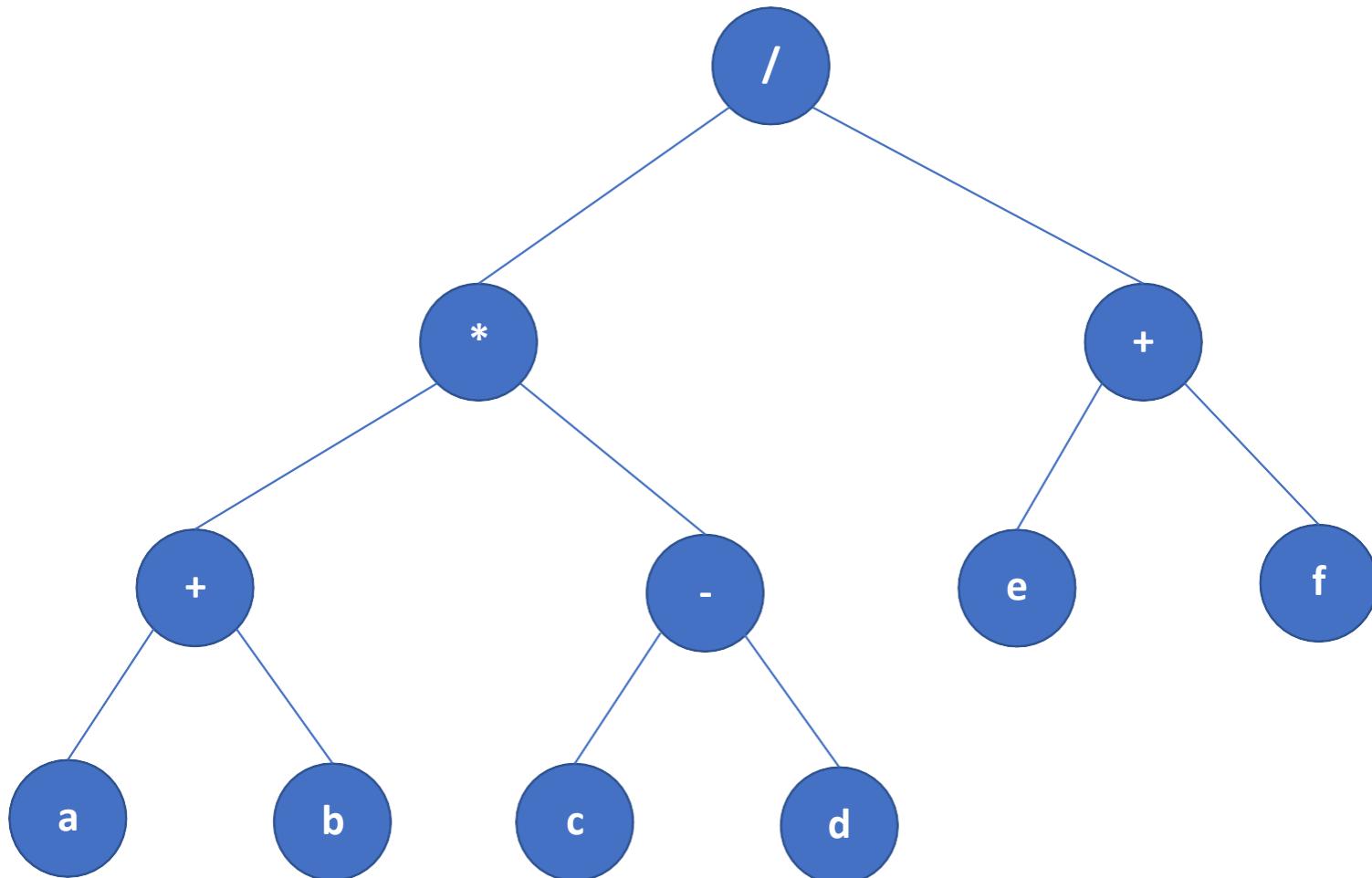


Contoh In-Order (Visit = print)



g d h b e i a f j c

In-Order Of Expression Tree



$a + b * c - d / e + f$

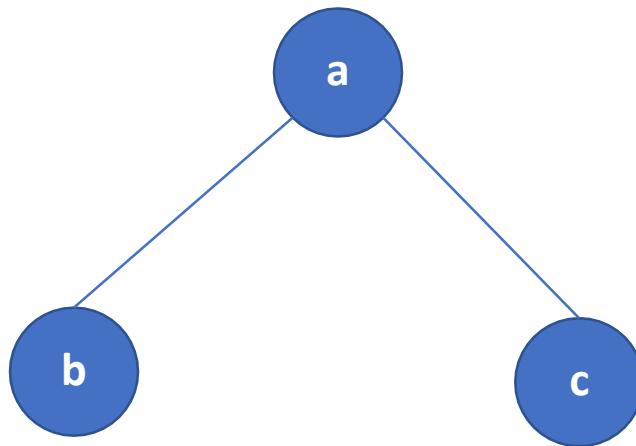
Gives Infix from of Expression!

Post-Order Traversal

□ Algoritma Post-Order traversal :

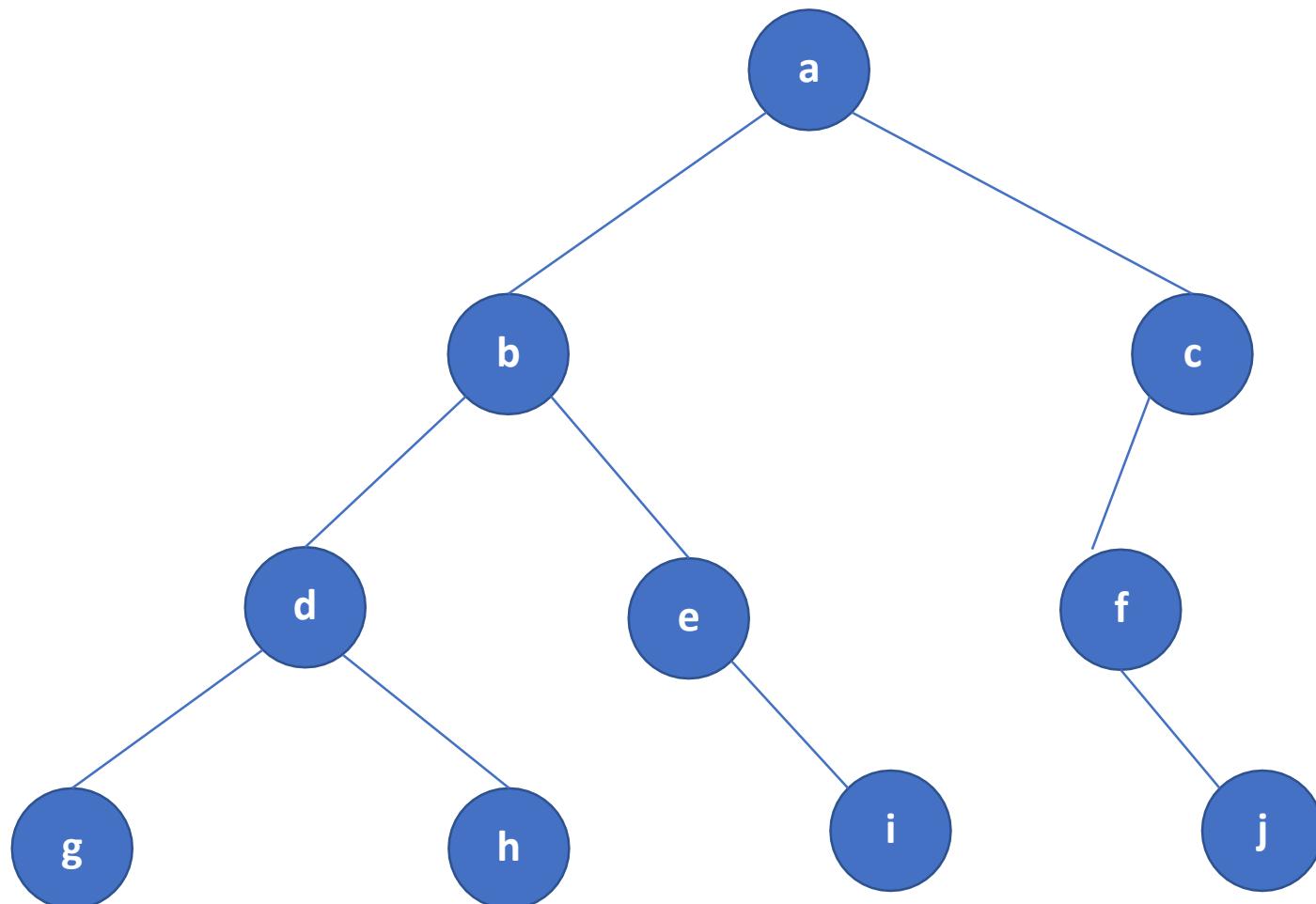
1. Secara rekursif mencetak seluruh data pada sub pohon
 ▶ **kiri**
2. Secara rekursif mencetak seluruh data pada sub pohon
 kanan
3. Cetak data pada **root**

Contoh Post-Order (Visit = print)

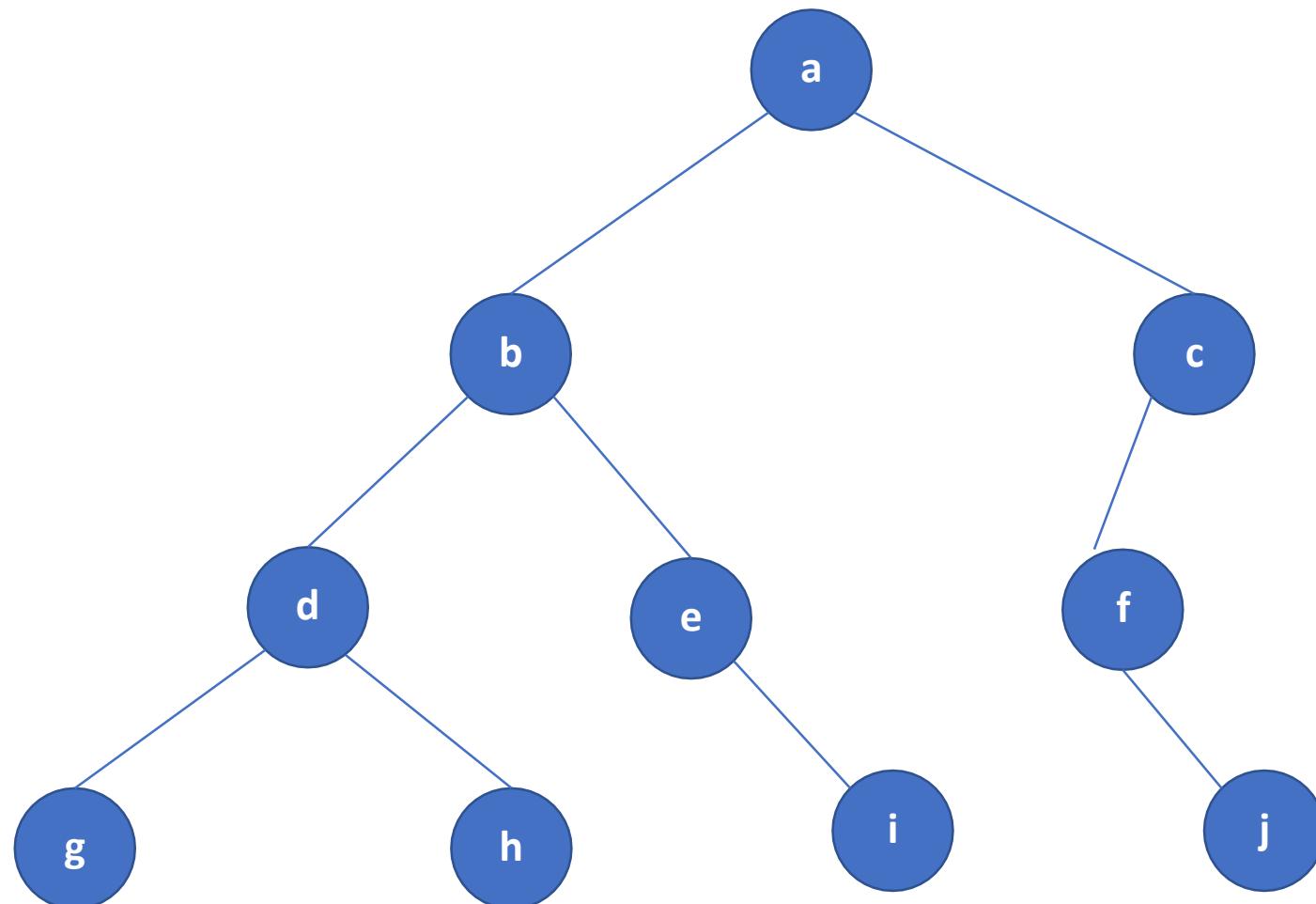


b c a

Contoh Post-Order (Visit = print)

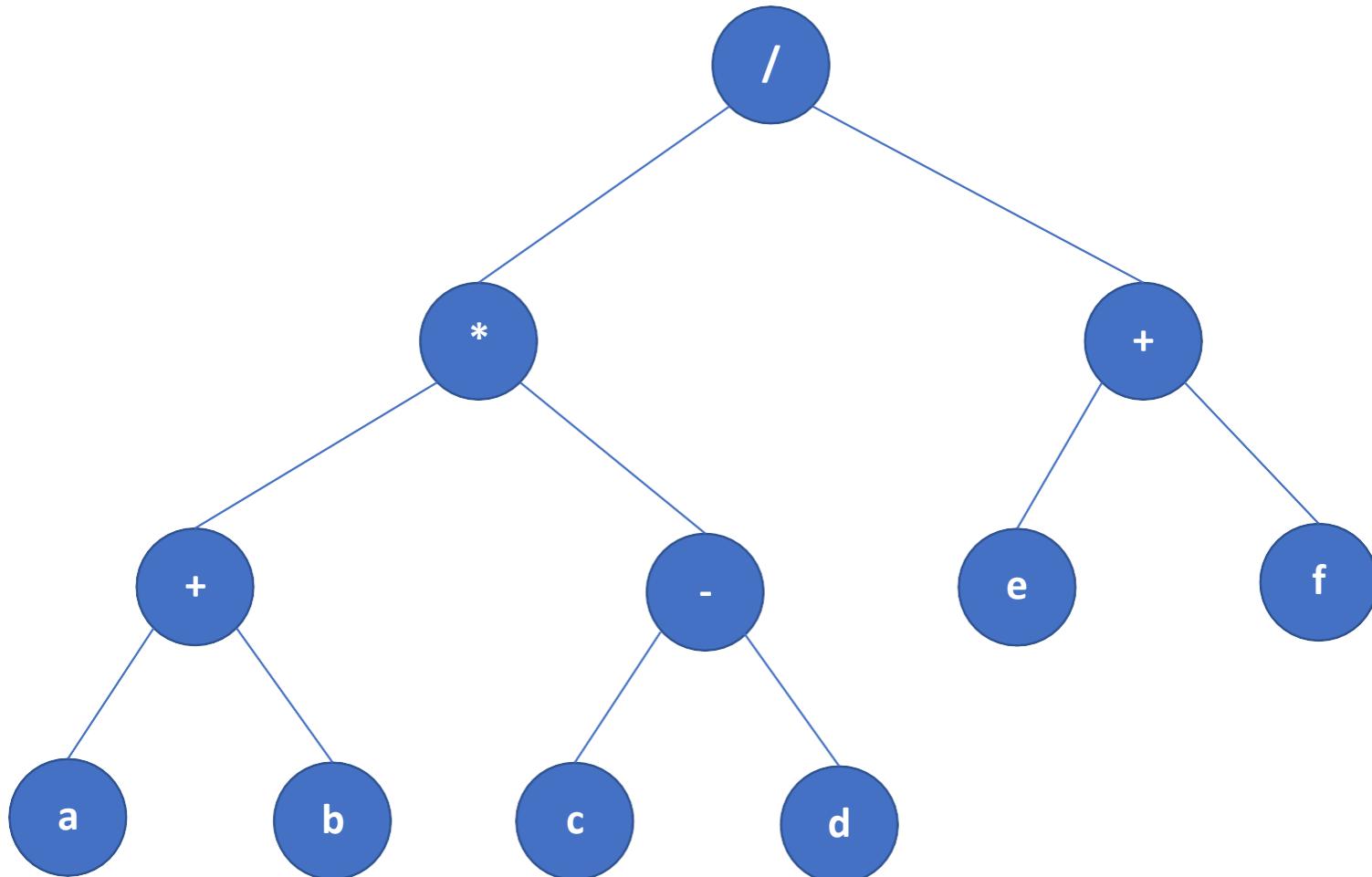


Contoh Post-Order (Visit = print)



g h d i e b j f c a

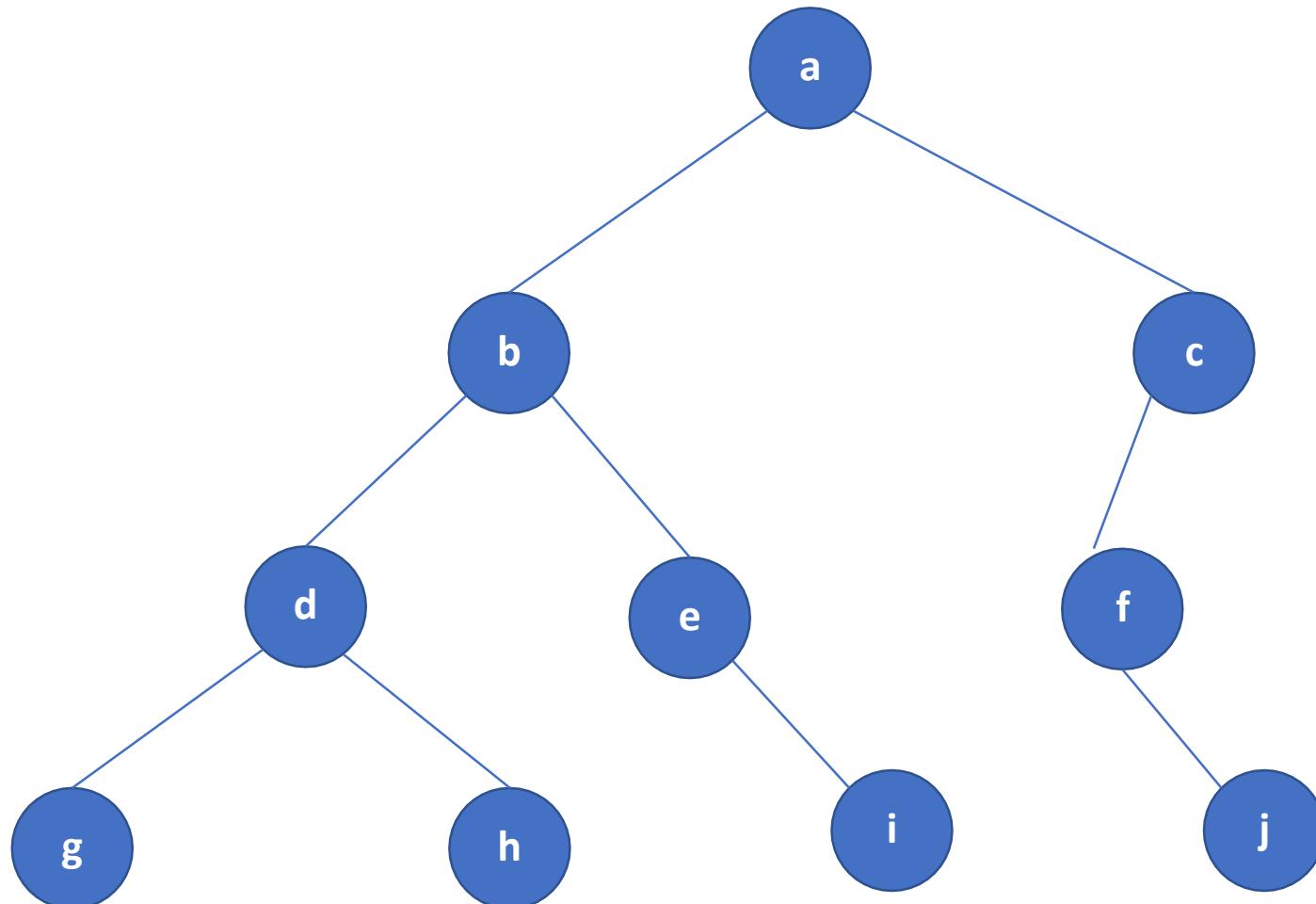
Post-Order Of Expression Tree



a b + c d - * e f + /

Gives postfix from of Expression!

Contoh Level-Order (Visit = print)



a b c d e f g h i j

Binary Search Tree (BST)

Binary Search Tree (BST)

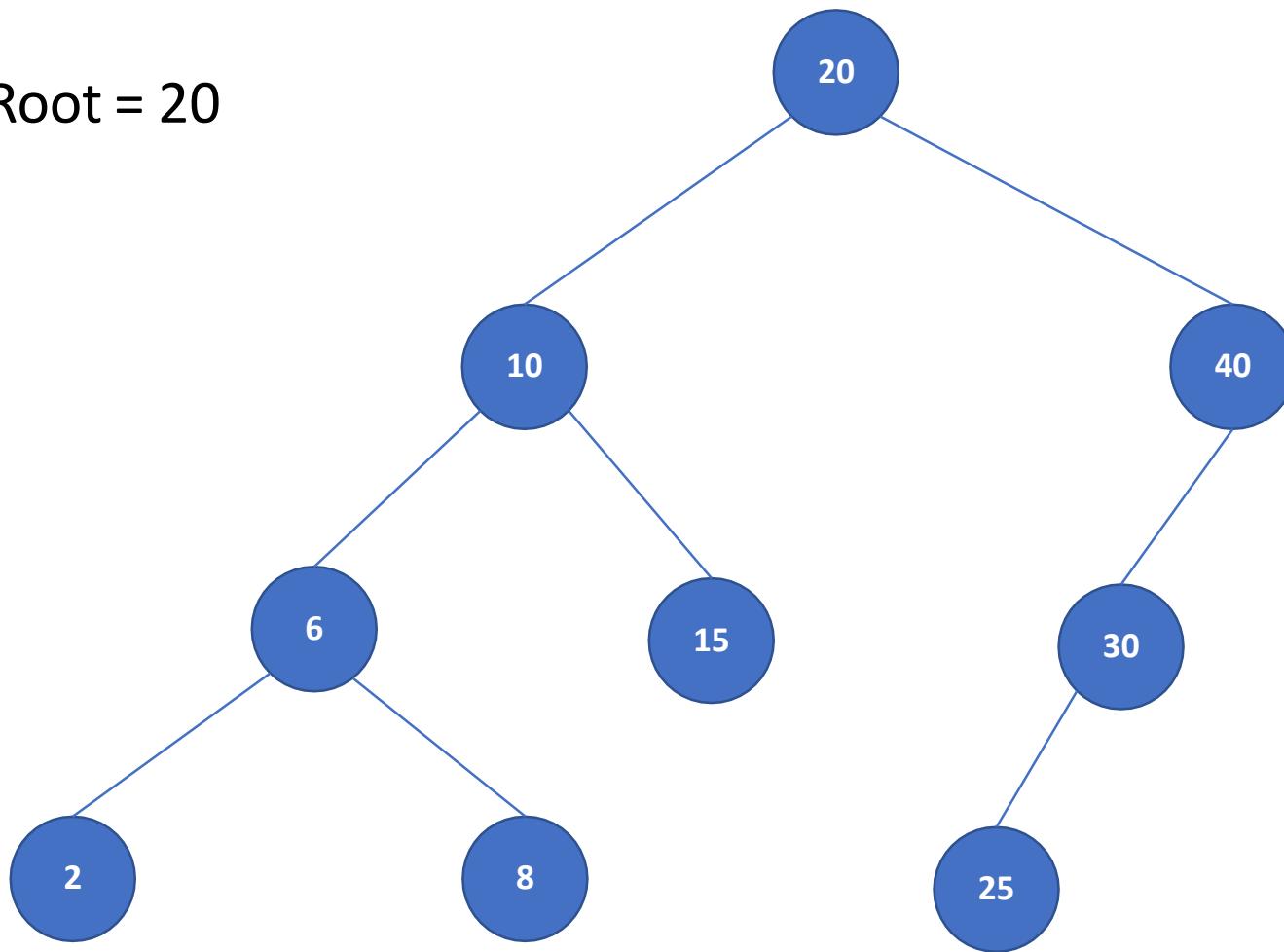
- Binary Search Tree disebut juga ***Ordered Binary Tree***
- Yaitu binary tree yang seluruh node-nya **terurut**.
- Aturan : data pada sub tree kiri lebih kecil dari data pada sub tree kanan.

Contoh Aplikasi BST

- ❑ Membangun daftar vocabulary yang merupakan bagian dari inverted index (sebuah struktur data yang digunakan oleh banyak mesin pencari seperti Google.com, Yahoo.com dan Ask.com)
- ❑ Banyak digunakan dalam bahasa pemrograman untuk mengelola dan membangun dynamic sets.

Contoh BST

Root = 20



Penambahan BST

Aturan MinMax :

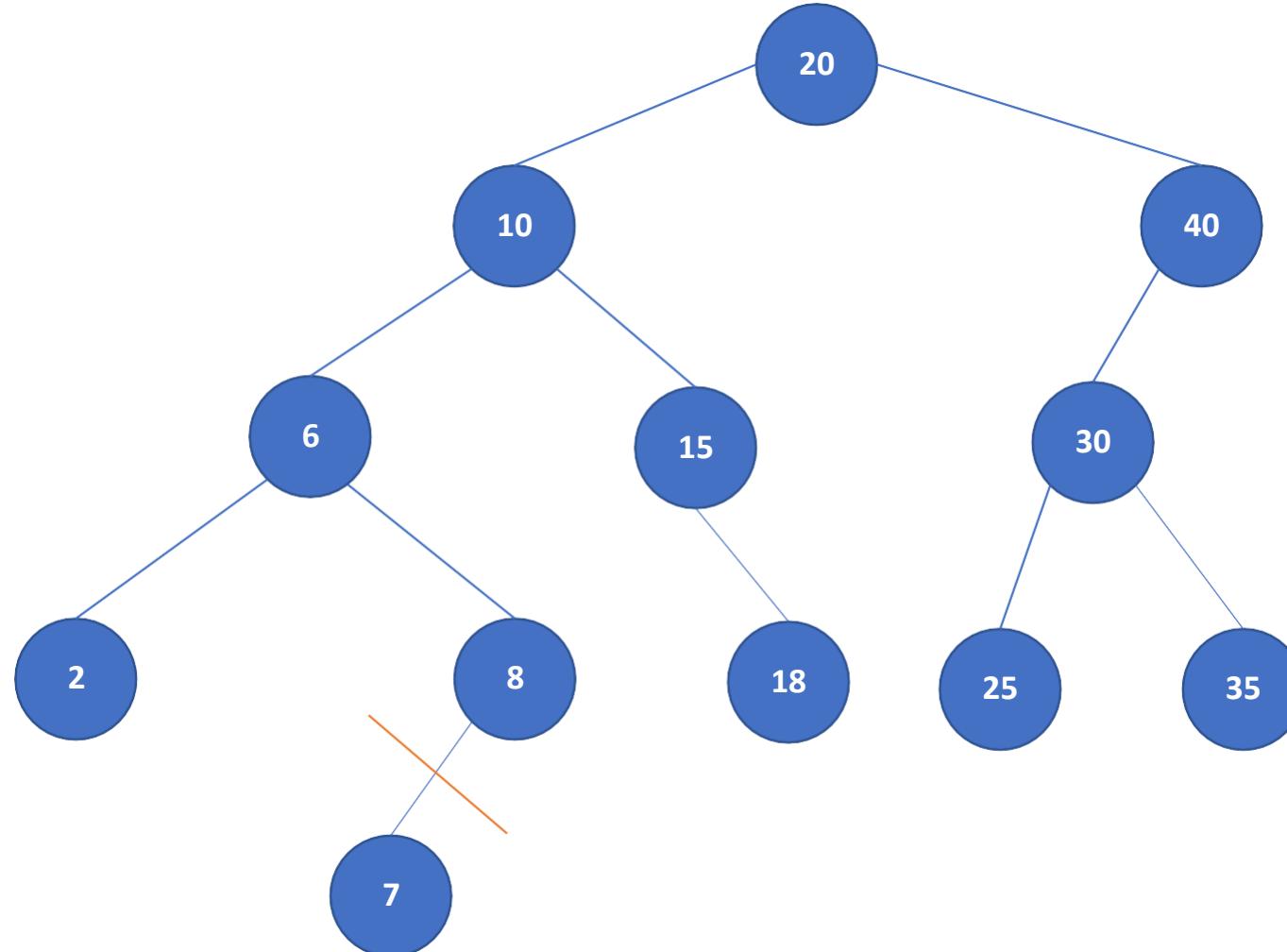
- ❑ Node yang bernilai **lebih kecil** dari root diletakkan pada sub tree **sebelah kiri**.
- ❑ Node yang bernilai **lebih besar** dari root diletakkan pada sub tree **sebelah kanan**.
- ❑ Jika ada nilai yang **sama** maka node tersebut di **overwrite**.

Penghapusan BST

Ada 3 kasus :

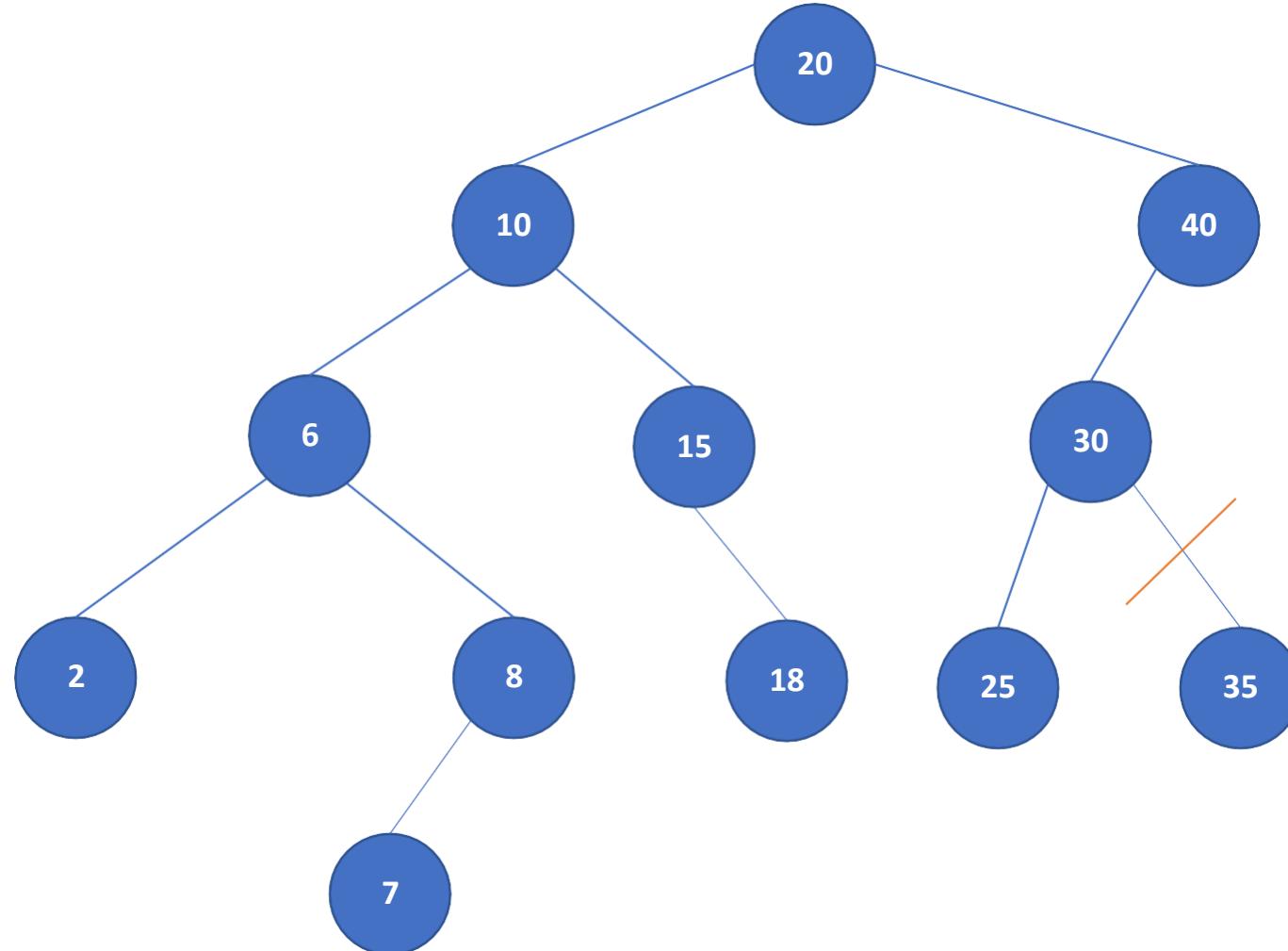
- Node adalah leaf/daun.
- Node memiliki degree 1
- Node memiliki degree 2

Penghapusan Node Daun (Node 7)



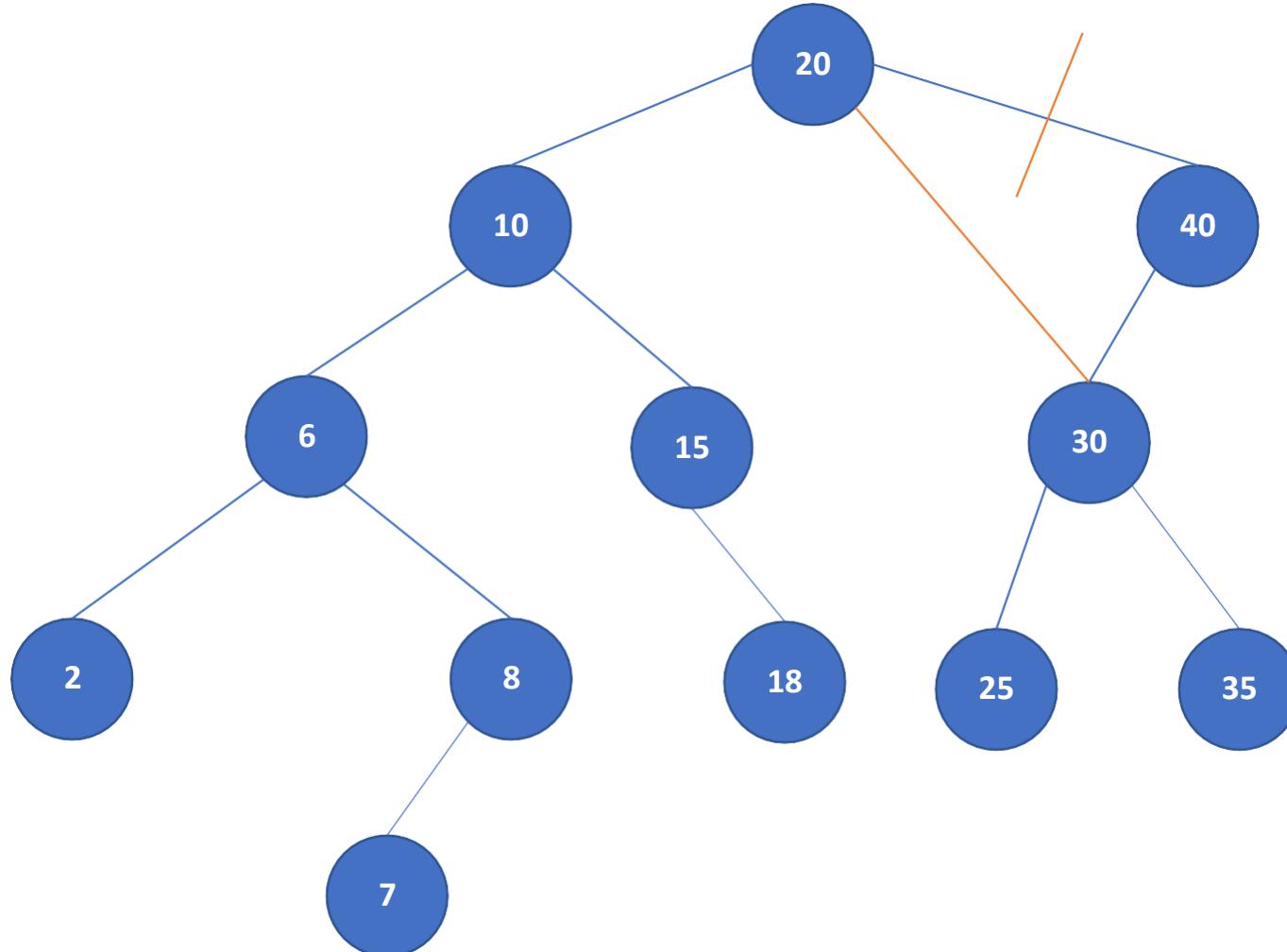
Remove a leaf element. Key = 7

Penghapusan Node Daun (Node 35)



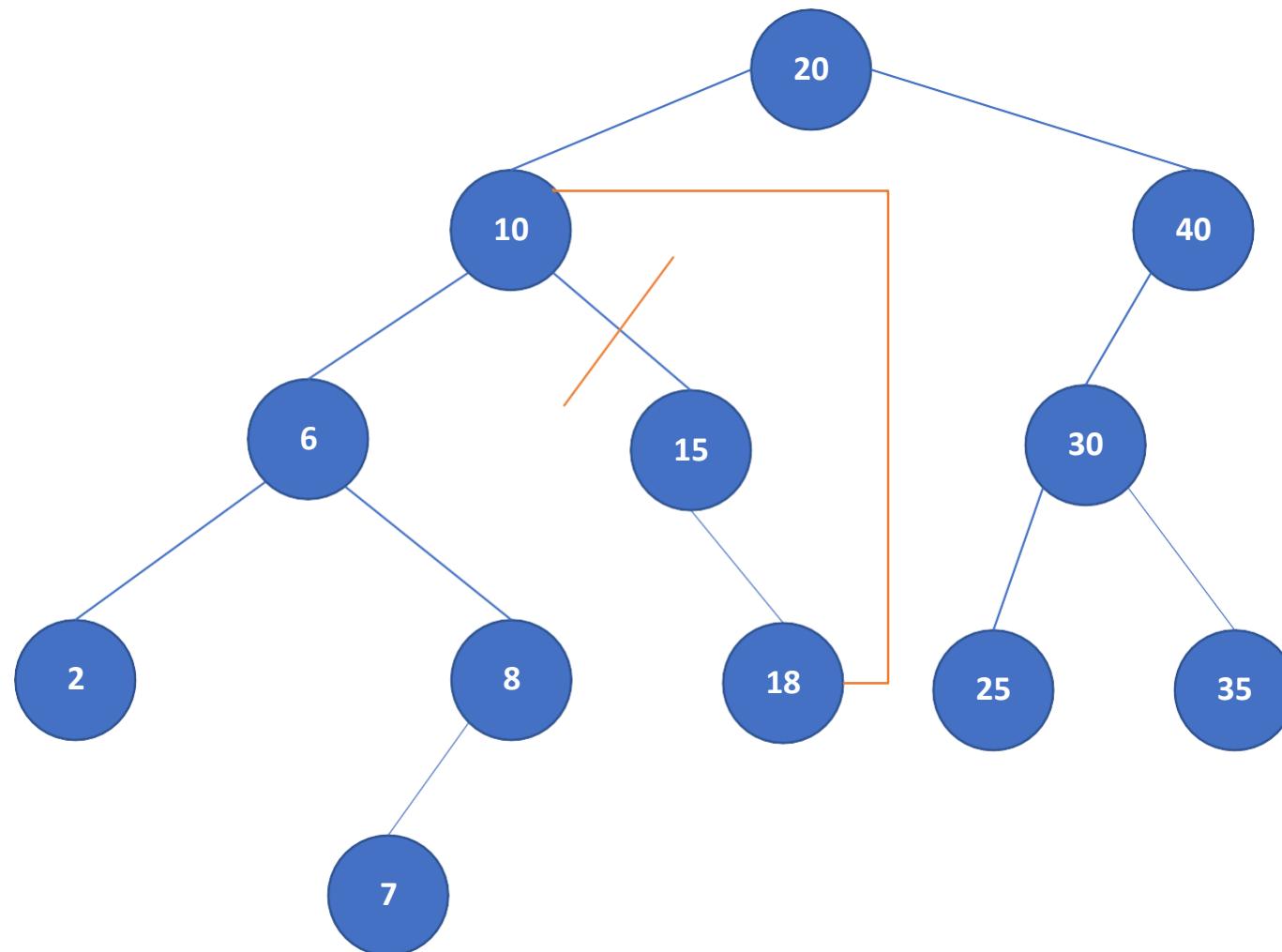
Remove a leaf element. Key = 35

Penghapusan Node Ber-degree 1 (Node 40)



Remove from a degree 1 node. Key = 40

Penghapusan Node Ber-degree 1 (Node 15)



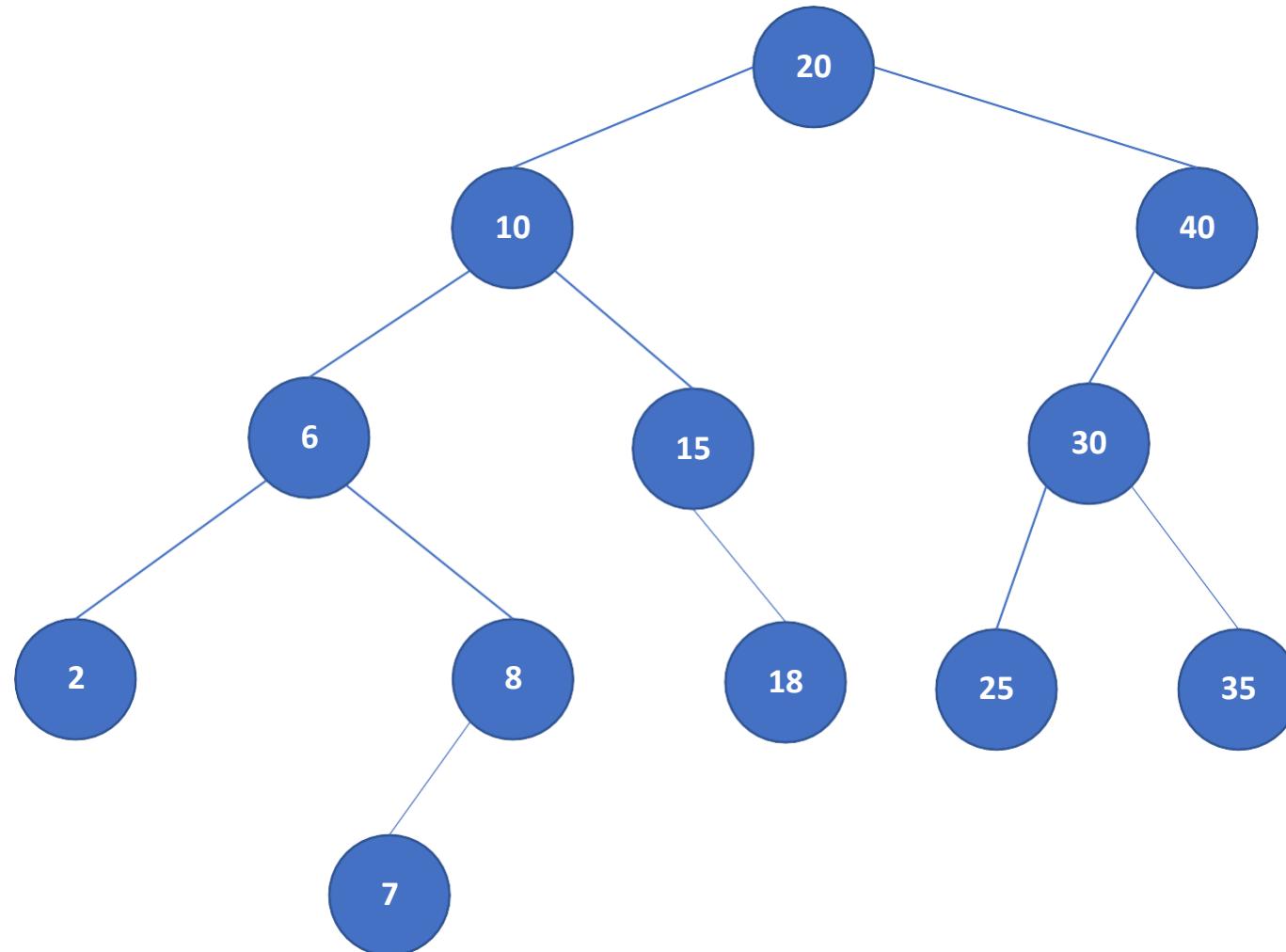
Remove from a degree 1 node. Key = 15

Aturan penghapusan node degree 2

Sebagai pengganti node yang dihapus dengan degree 2, aturannya sebagai berikut :

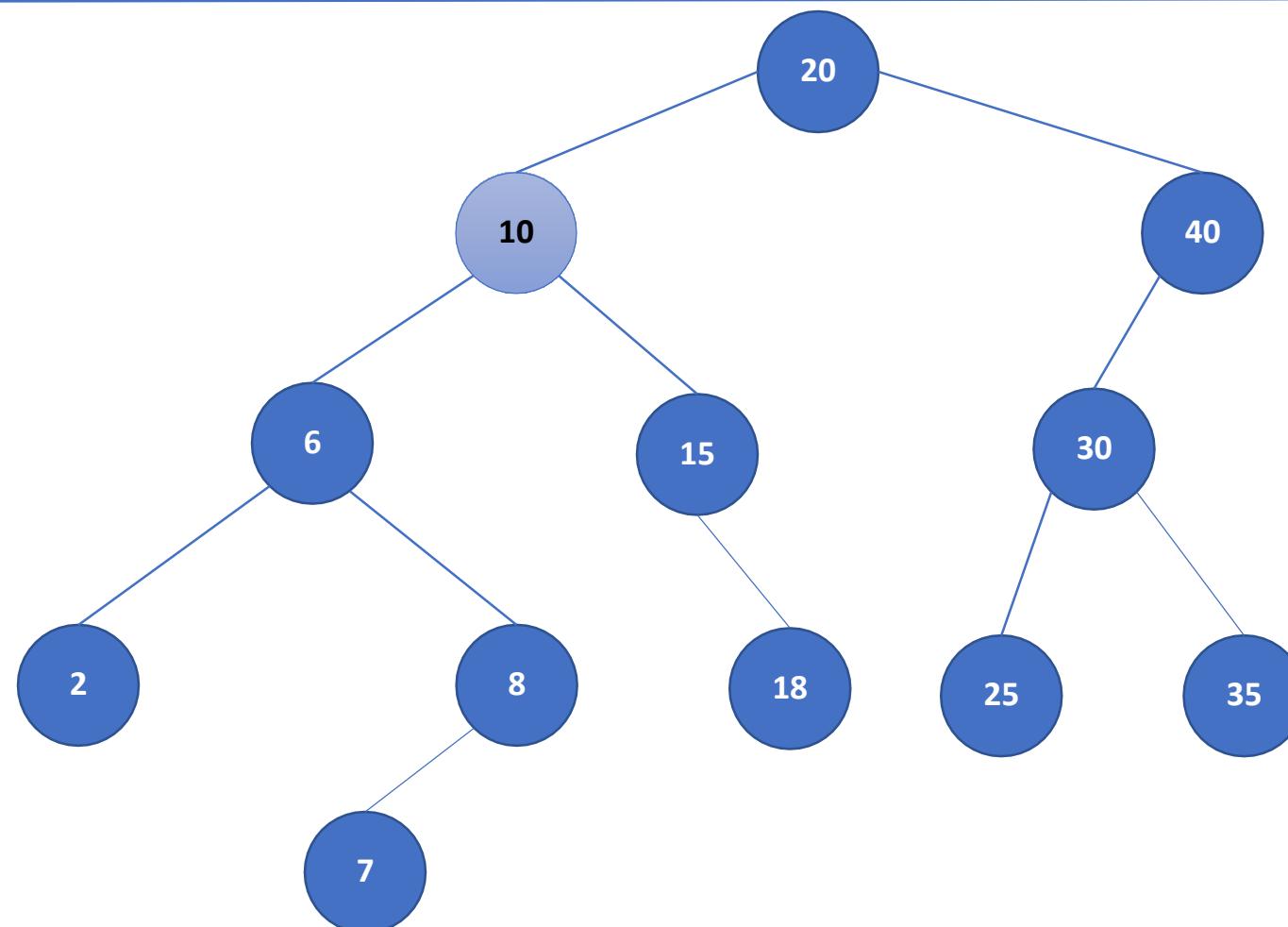
- Ambil node paling besar dari sub tree kiri, atau
- Ambil node paling kecil dari sub tree kanan

Penghapusan Node Ber-degree 2 (Node 10)



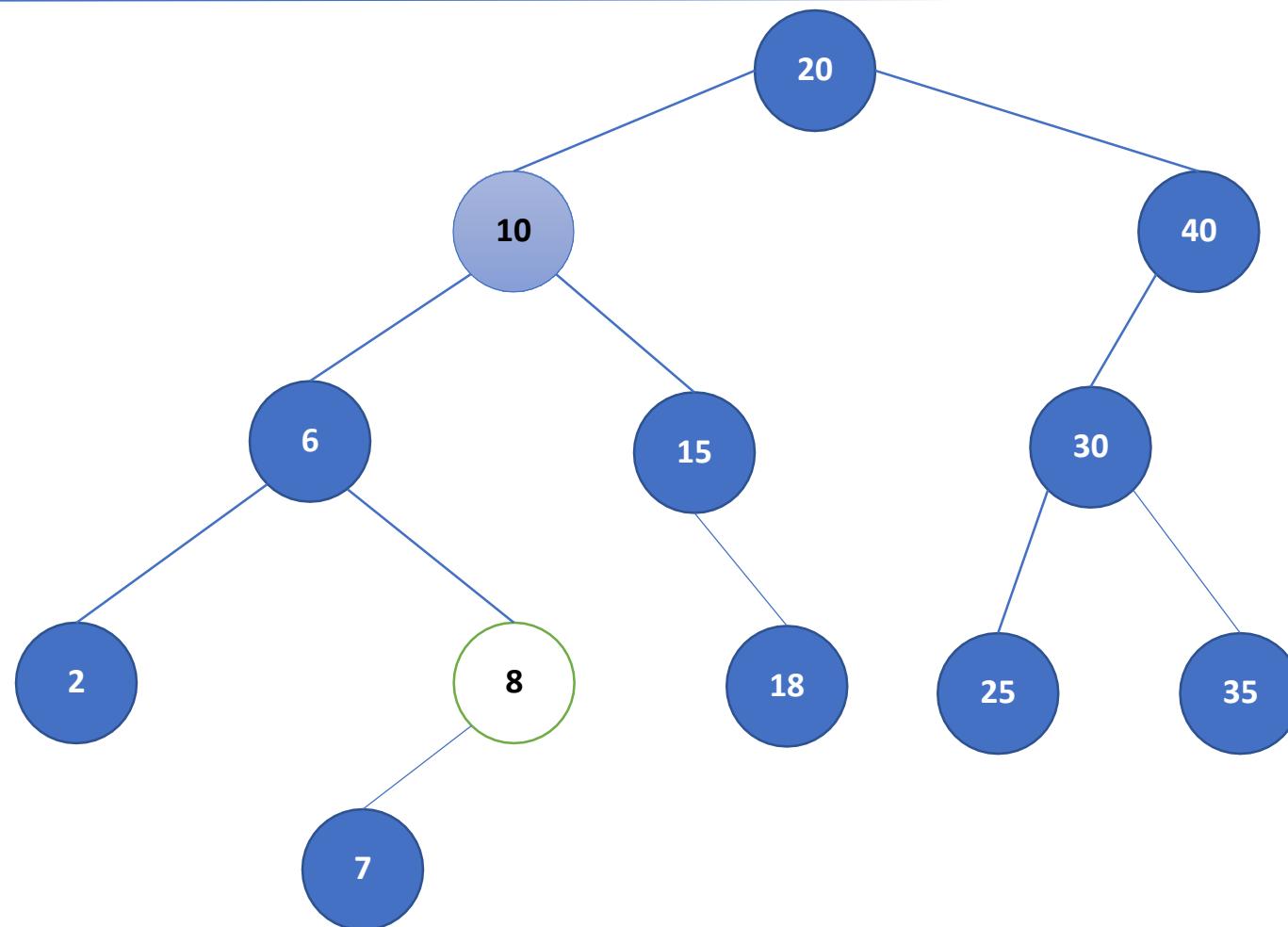
Remove from a degree 2 node. Key = 10

Penghapusan Node Ber-degree 2 (Node 10)



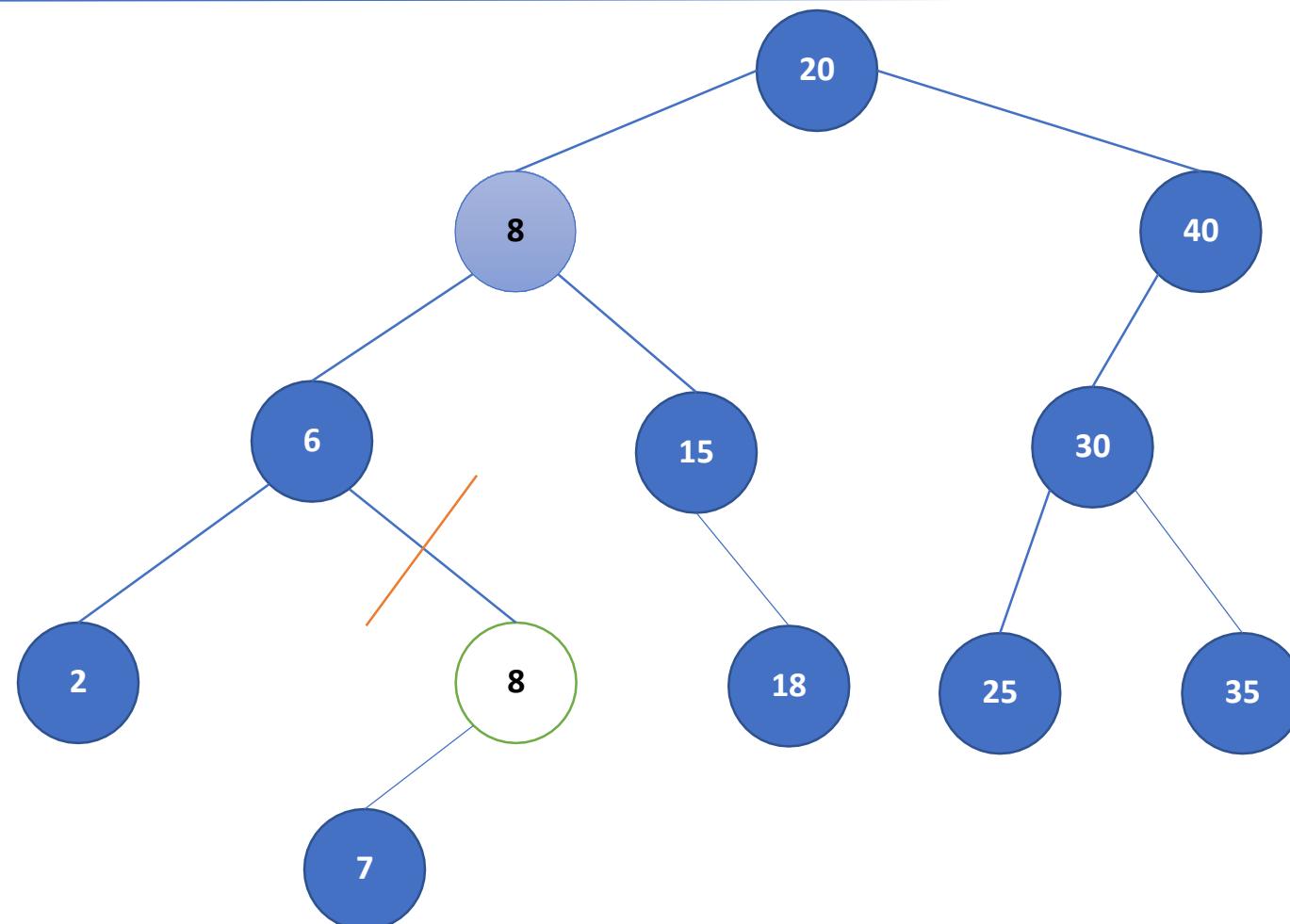
Replace with largest key in left sub tree (or smallest in right sub tree.)

Penghapusan Node Ber-degree 2 (Node 10)



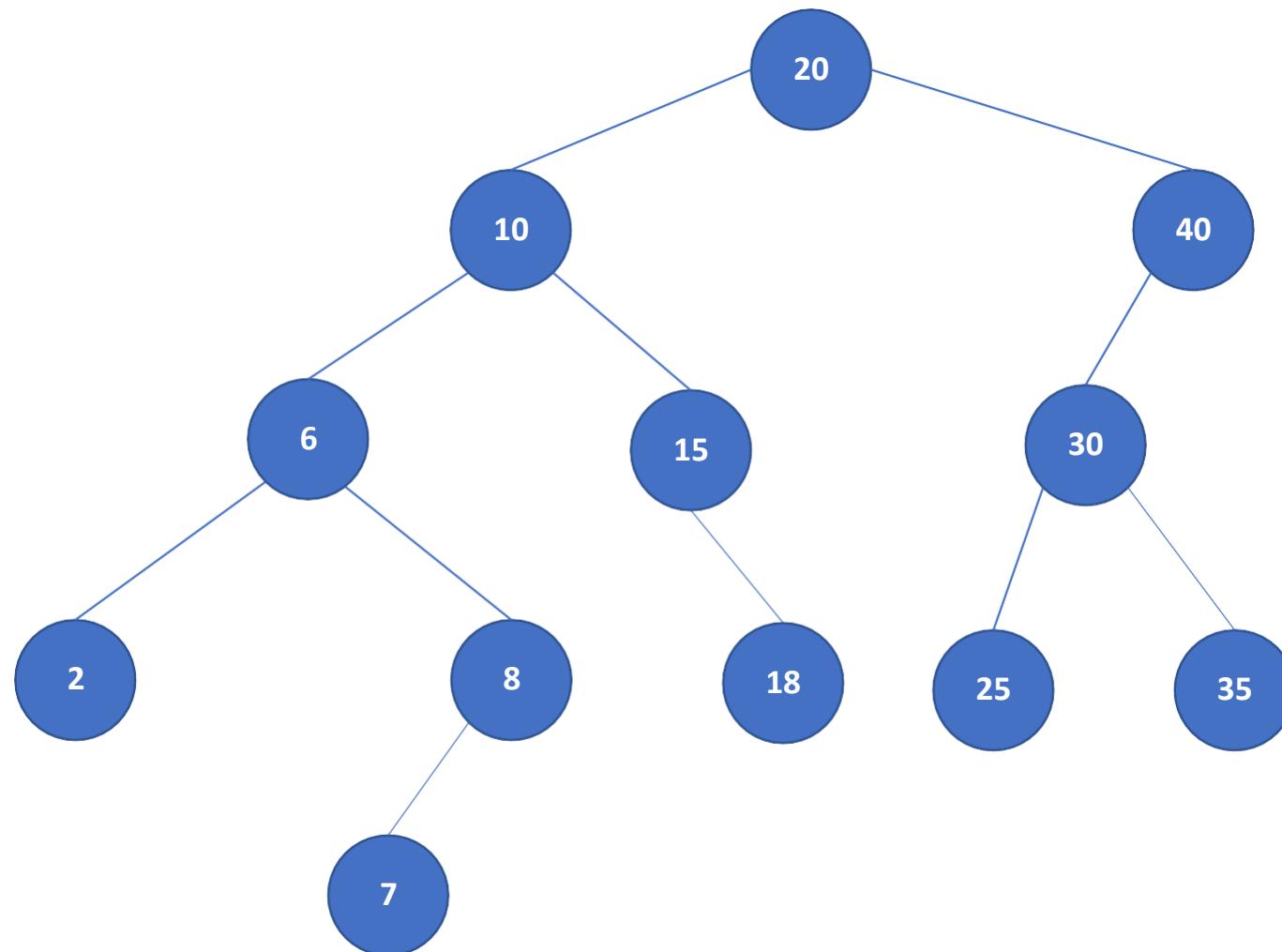
Replace with largest key in left sub tree (or smallest in right sub tree.)

Penghapusan Node Ber-degree 2 (Node 10)



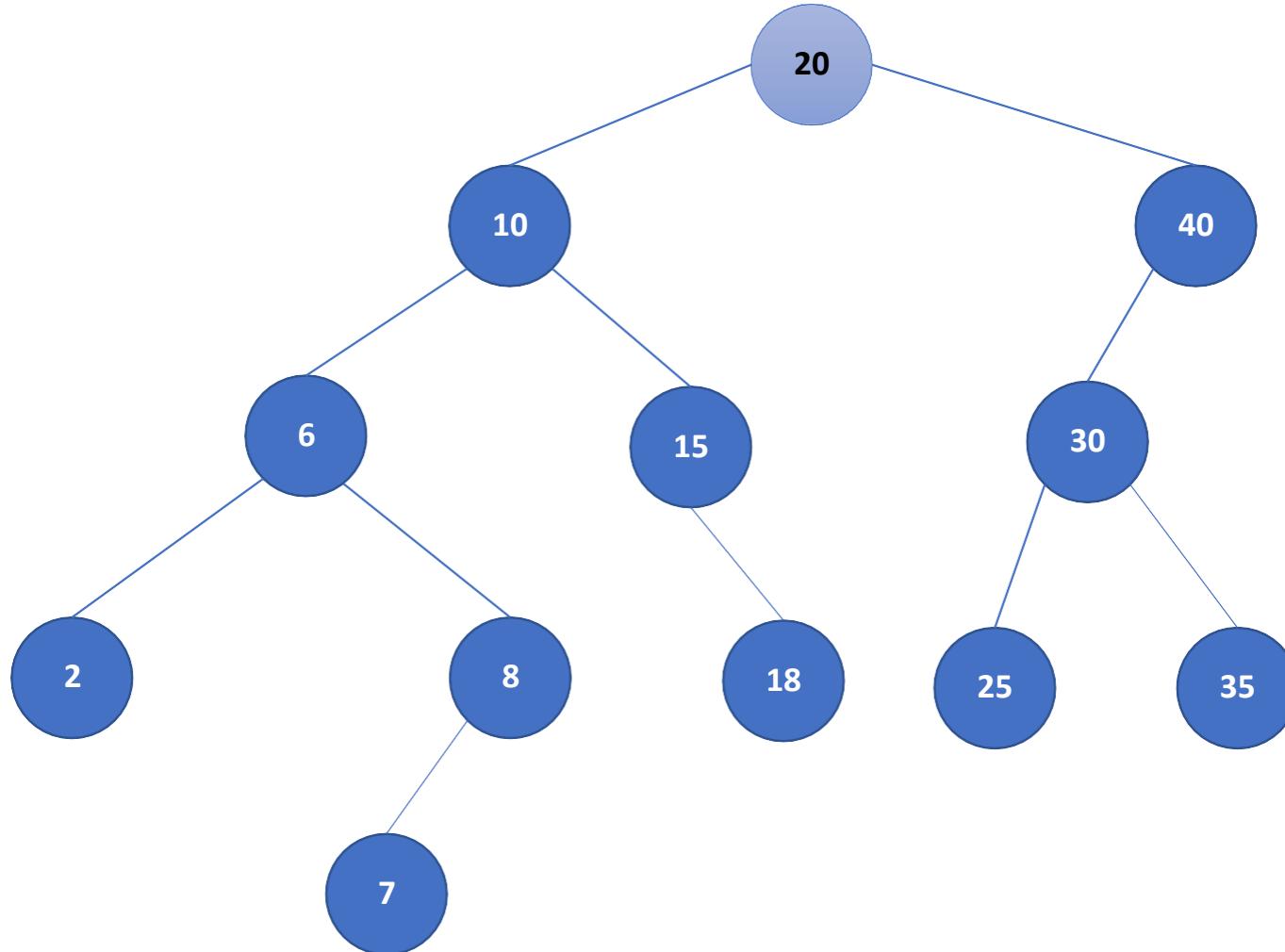
Replace with largest key in left sub tree (or smallest in right sub tree.)

Contoh Latihan



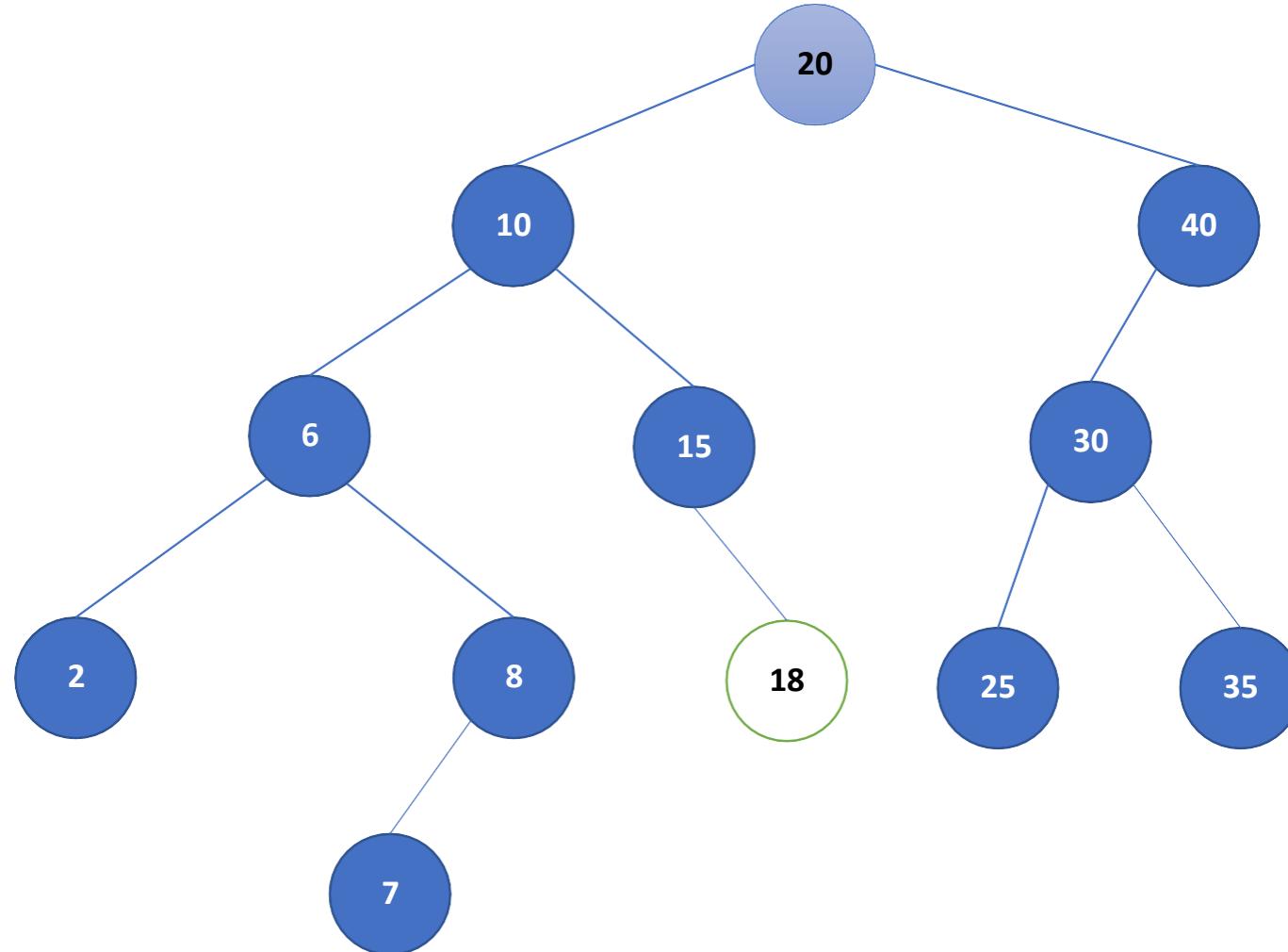
Remove from a degree 2 node. Key = 20

Penghapusan Node Ber-degree 2



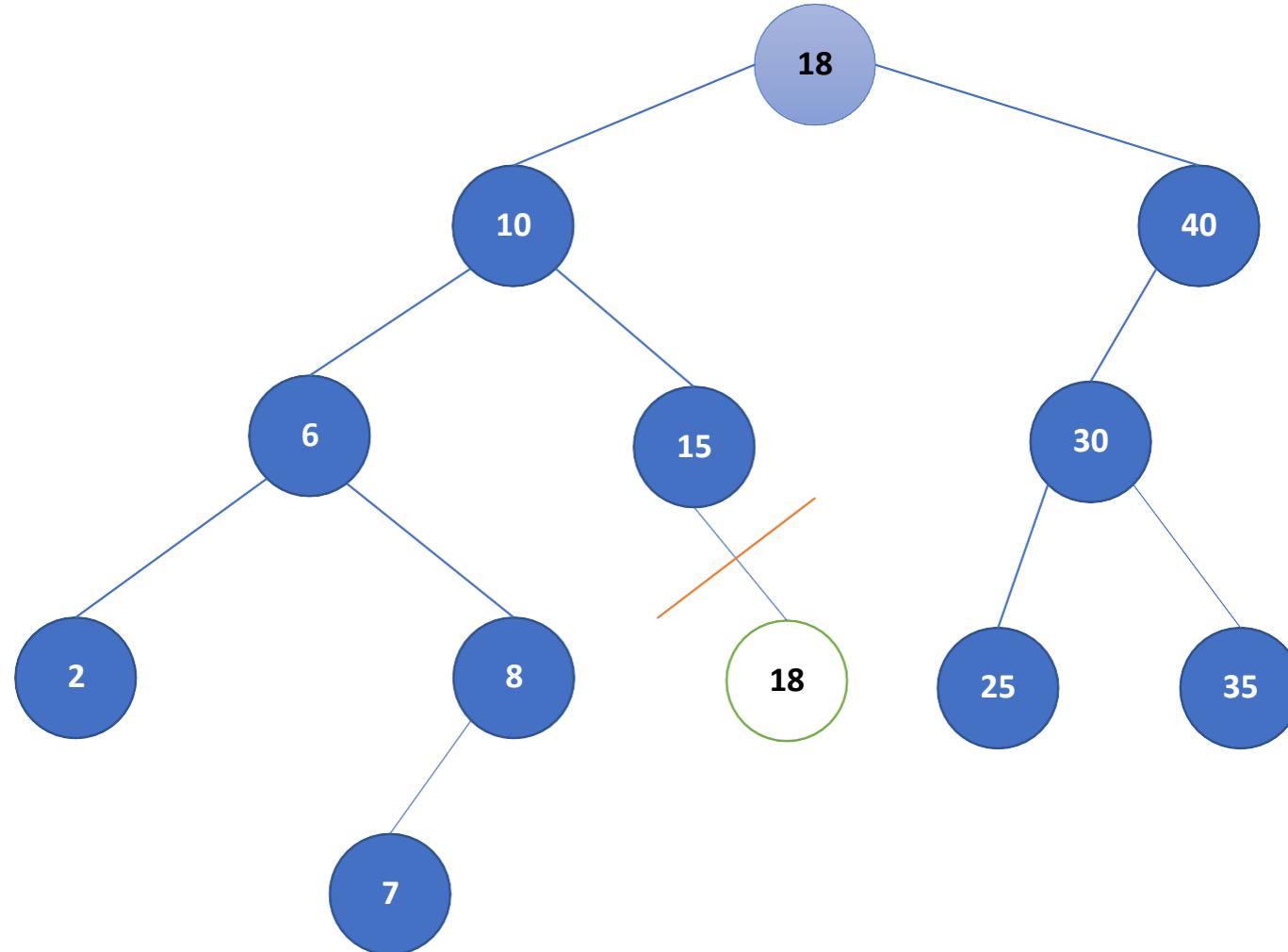
Replace with largest in left sub tree

Penghapusan Node Ber-degree 2



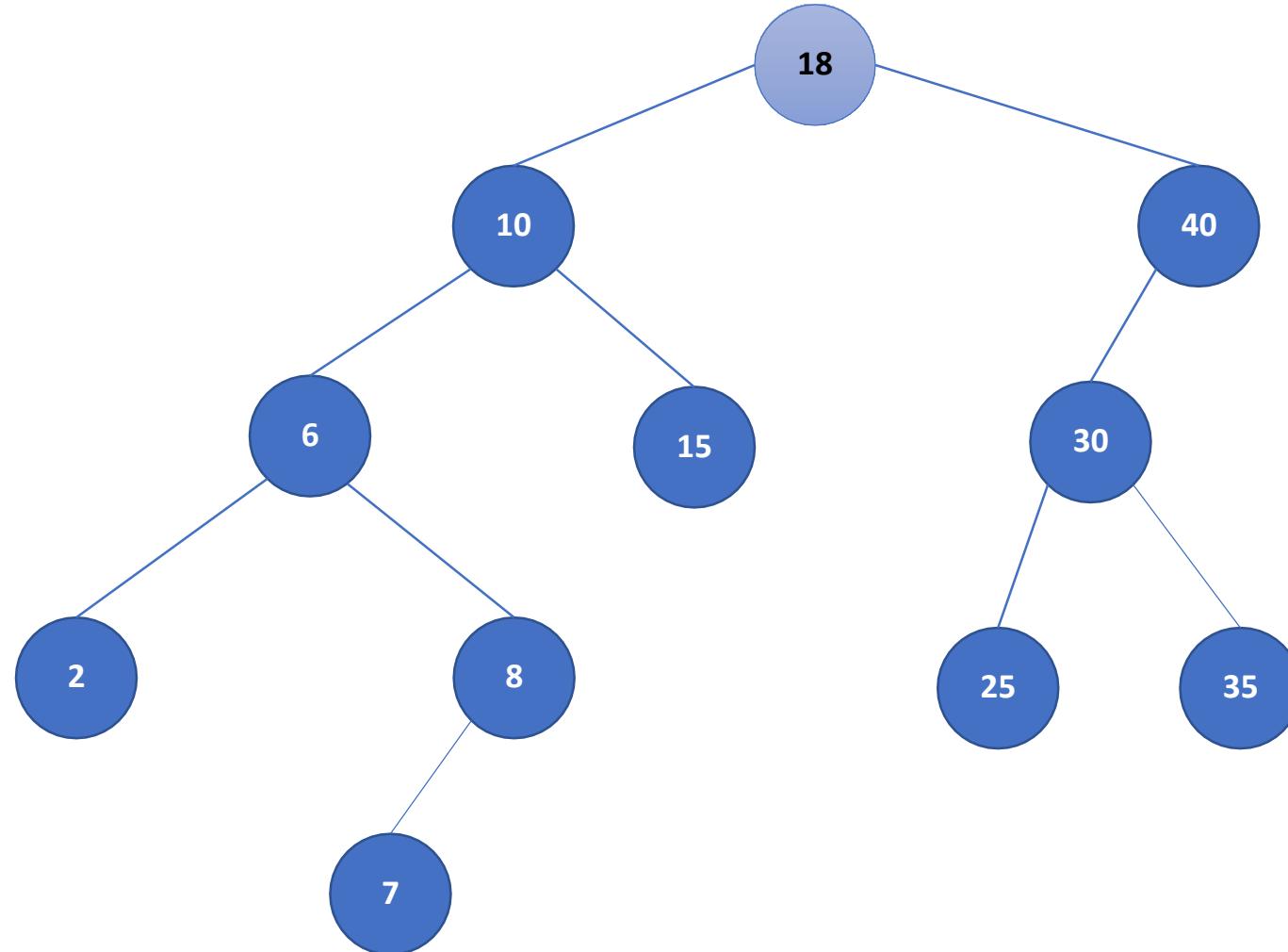
Replace with largest in left sub tree

Penghapusan Node Ber-degree 2



Replace with largest in left sub tree

Hasil Akhir



Implementasi Binary Search Tree

Operasi-operasi yang dilakukan pada binary search tree meliputi :

1. Penambahan node
2. Penghapusan node
3. Pencarian node

Contoh Algoritma

□ Penghapusan :

```
Node2P remove(int x, Node2P t)
{
    if (t == null) t=null;
    if (x < t.data) {
        t.previous = remove(x, t.previous);
    } else if (x > t.data) {
        t.next = remove(x, t.next);
    } else if (t.previous != null && t.next != null) {
        t.data = findMin(t.next).data;
        t.next = removeMin(t.next);
    } else {
        t = (t.previous != null) ? t.previous : t.next;
    }
    return t;
}
```

Contoh Algoritma

- ❑ Penghapusan node terkecil :

```
Node2P removeMin(Node2P t)
{
    if (t == null) t=null;

    if (t.previous != null) {
        t.previous = removeMin (t.previous);
    } else {
        t = t.next;
    }
    return t;
}
```

Contoh Algoritma

- ❑ Pencarian node terkecil :

```
Node2P findMin (Node2P t)
{
    if (t == null) t=null;

    while (t.previous != null) {
        t = t.previous;
    }
    return t;
}
```



Terima
kasih