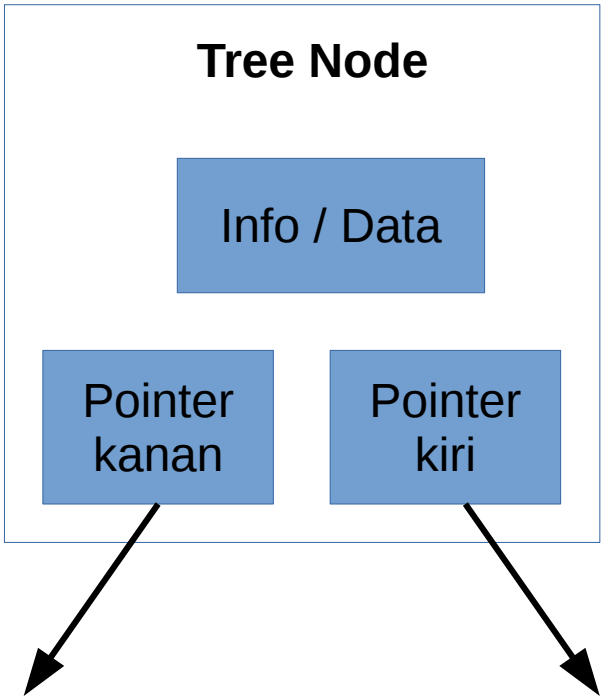
Implementasi Tree

Membangun tree (pohon biner)

Implementasi tree node

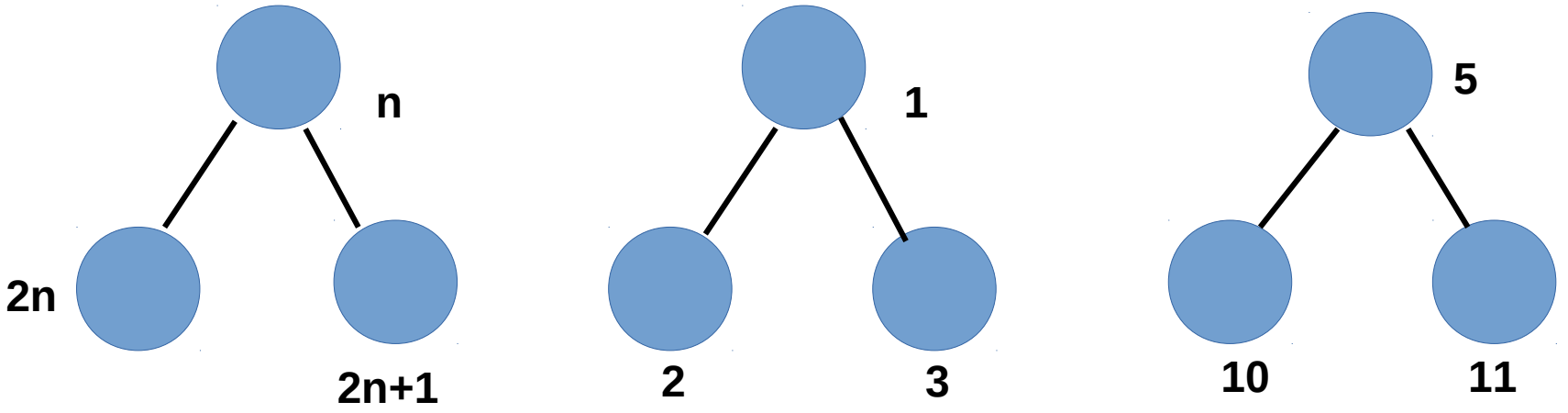
- Tree node secara simpel terdiri dari 2 pointer

```
template <class Item>
class binary_tree_node
{
public:
    || Public member functions will give access to the data and links.
private:
    Item data_field;
    binary_tree_node *left_field;
    binary_tree_node *right_field;
};
```



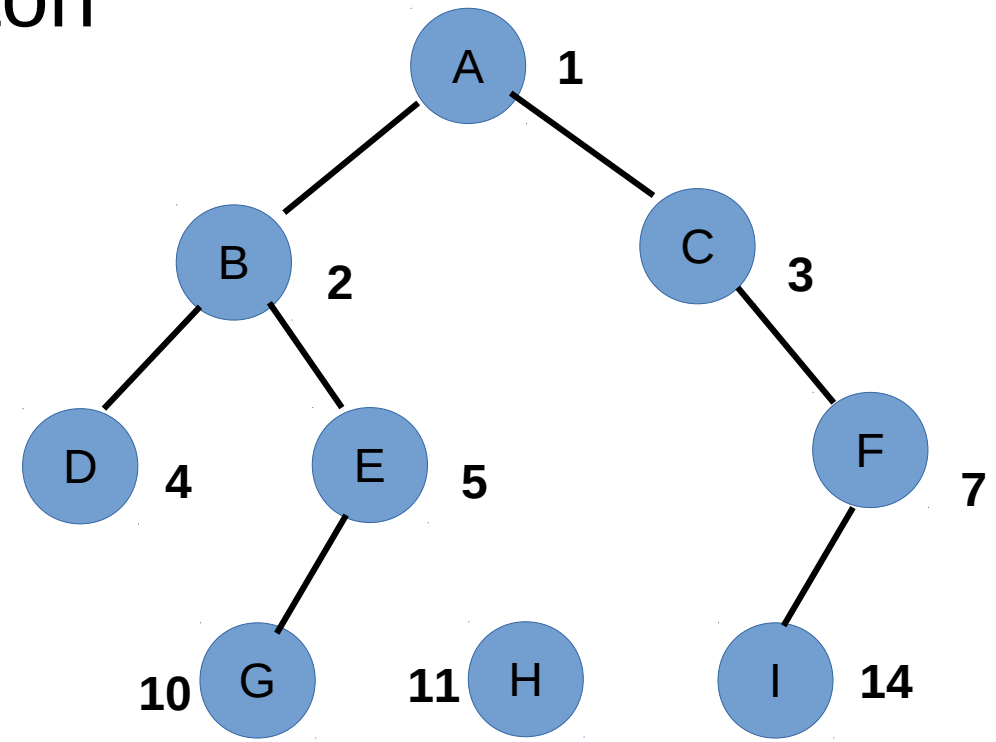
Penomoran simpul

- Simpul pada pohon perlu diberi nomor untuk memudahkan dalam representasi data
- Berdasar konvensi (kesepakatan), bila sebuah simpul bernomor n maka sub ordinat (anak) kiri akan bernomor 2n dan sub ordinat kanan bernomor 2n + 1



Penomoran simpul

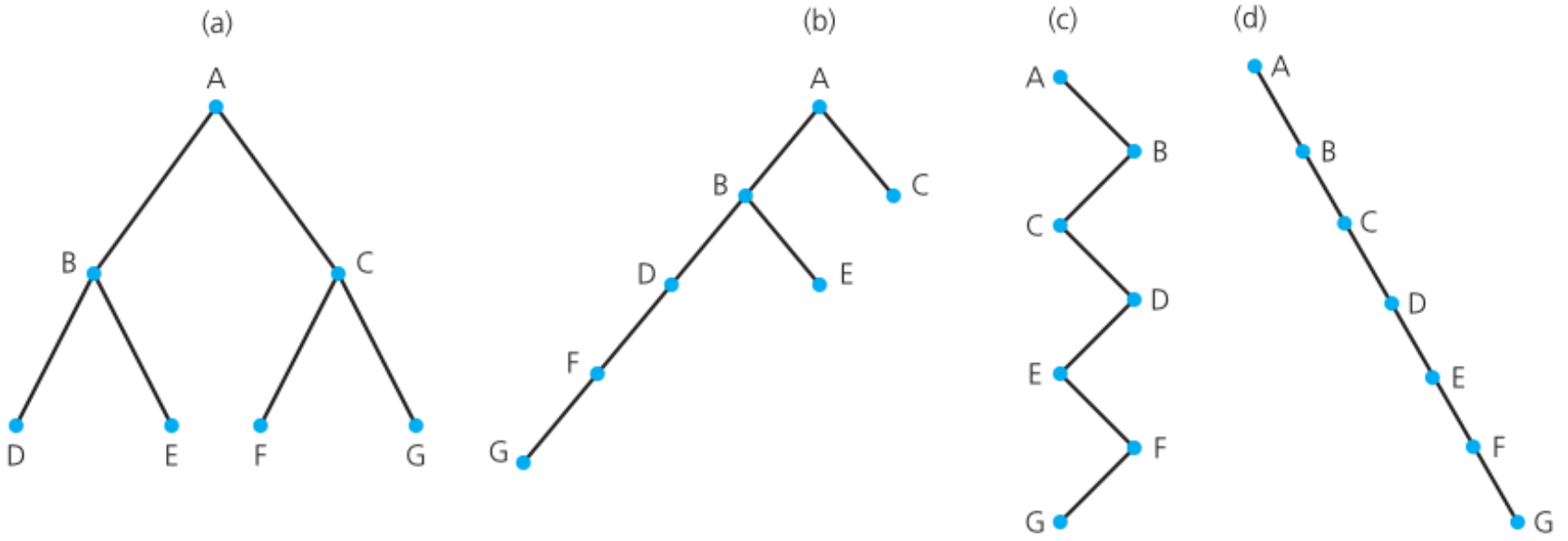
- contoh



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	D	E		F			G	H			I	

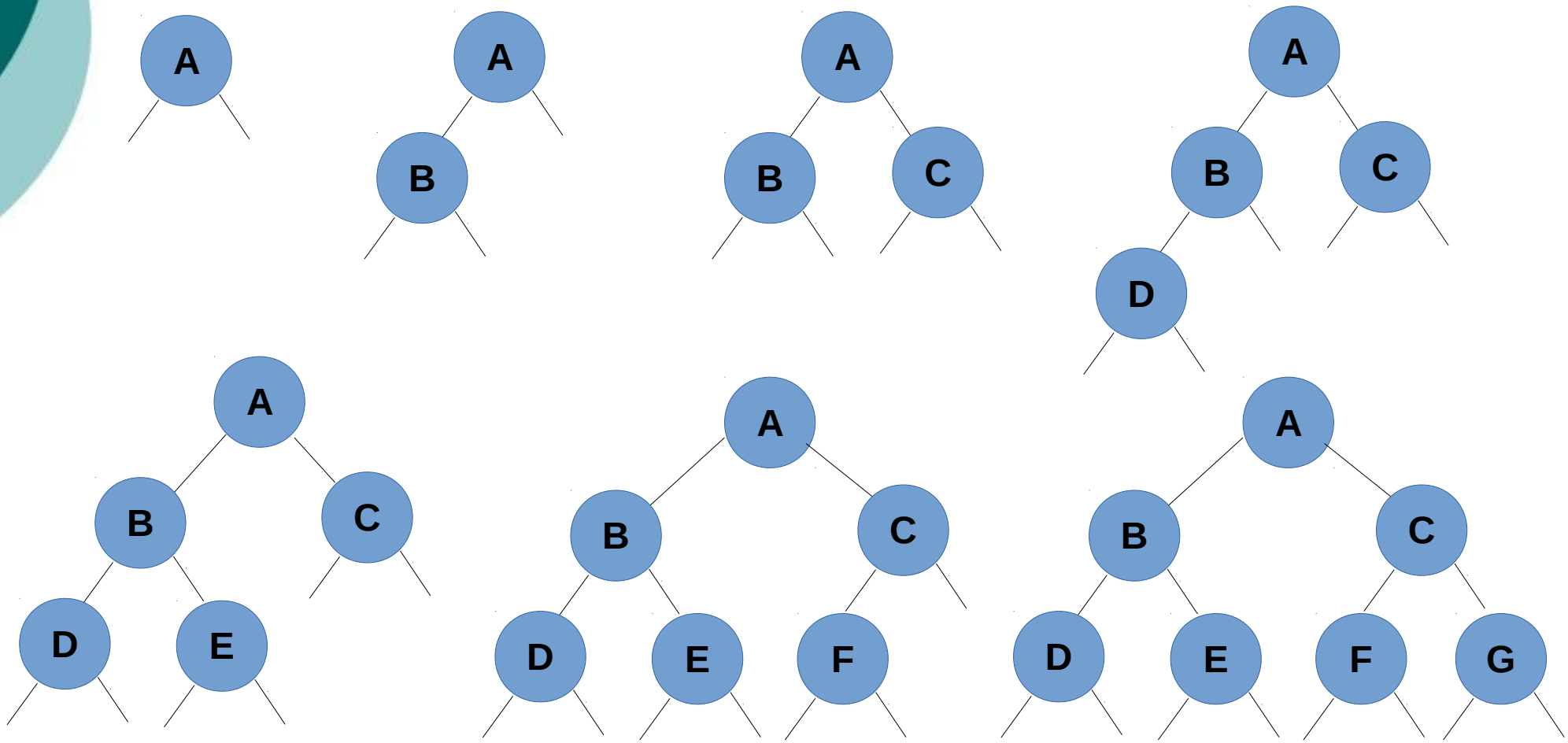
Algoritma menambah node

- Struktur data tree tidak mengatur secara spesifik dan ketat, ketika kita menambah node ke dalam tree, dimana node baru itu diletakkan
- Posisinya bebas tergantung pohon seperti apa yang akan kita buat



Insert sesuai nomor urut simpul

- Mulai insert node pertama sebagai root

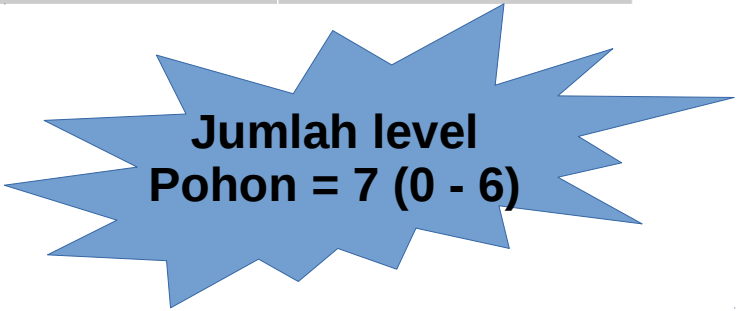


Insert simpul sesuai nomor

- Contoh: kita akan membuat pohon dengan menyisipkan node sesuai nomor urut
- Diketahui jumlah node yang akan dimasukkan dalam pohon adalah 127

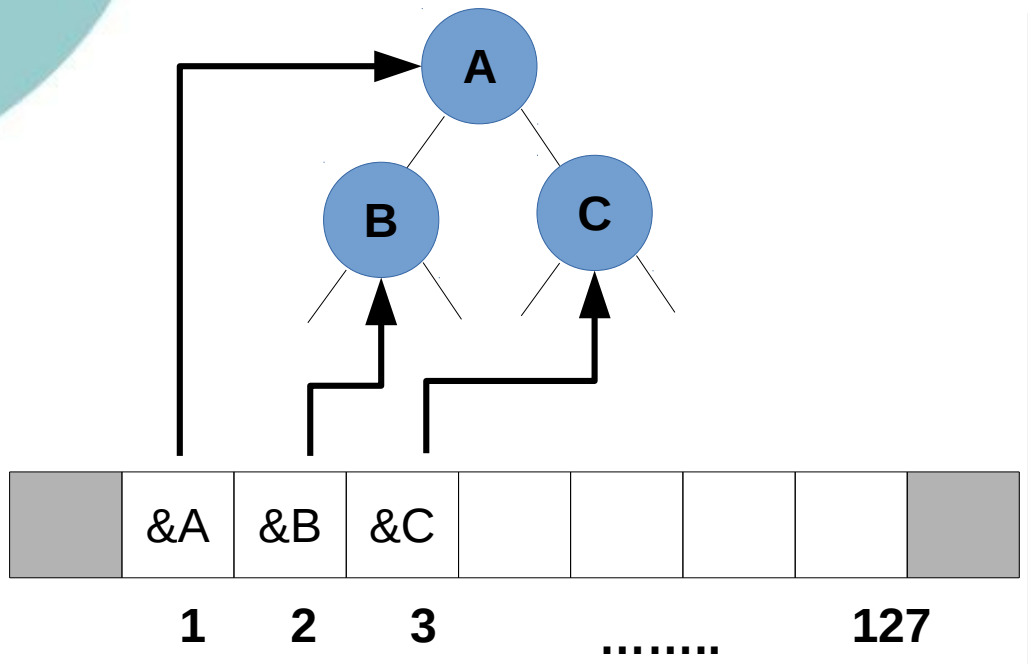
Level	Max nodes
0	1
1	2
2	4
3	8
4	16

Level	Max nodes
5	32
6	64
Total	**127**



Implementasi insert

- Maka kita buat **array of pointer** yang nantinya akan mengalokasi memori untuk membuat node



pseudocode

```
X = new Node()     //alokasi memori dinamis
set X sebagai root  //variabel Root = X
i = 1, j = 1
set Q[i] = Root    //Q adalah array of pointer

while i<127        //selama masih ada node
   set current = Q[i]
   P = new Node()     //alokasi memori dinamis
   current->left = P //jadikan anak kiri
   j++
   Q[j] = P           //Q[2] menunjuk B

   P = new Node()
   current->right = P
   j++
   Q[j] = P           //Q[3] menunjuk C

   i++
```

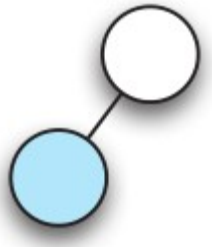
Insert simpul agar pohon balance

- Algoritma menambah node agar pohon balance
- Pengertian balance: tinggi anak kiri (subtree) dan anak kanan sama (subtree)
- Jika **pohon kosong** maka node yang baru jadi root
- Jika anak kiri (subtree kiri) lebih tinggi dari anak kanan (subtree kanan) maka insert ke subtree kanan
- Jika yang kanan lebih tinggi, insert ke subtree kiri

Kondisi inisial: pohon kosong

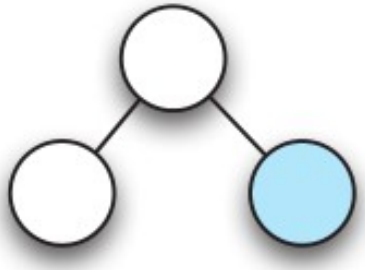
- Add node ke pohon kosong: node baru jadi root

- **Add lagi node baru**, apakah kiri > kanan ?
  Tidak karena anak kiri dan kanan masih kosong
- Maka insert ke subtree kiri

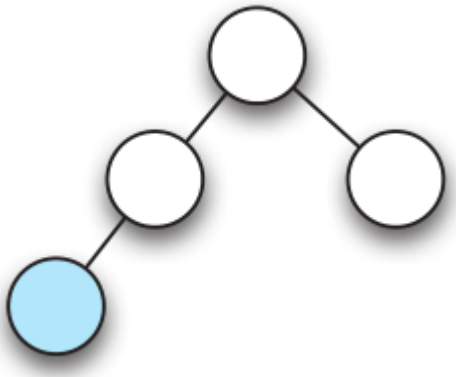


Add node (3)

- **Add lagi node baru** : subtree kiri lebih tinggi dari kanan? Ya maka insert ke sebelah kanan

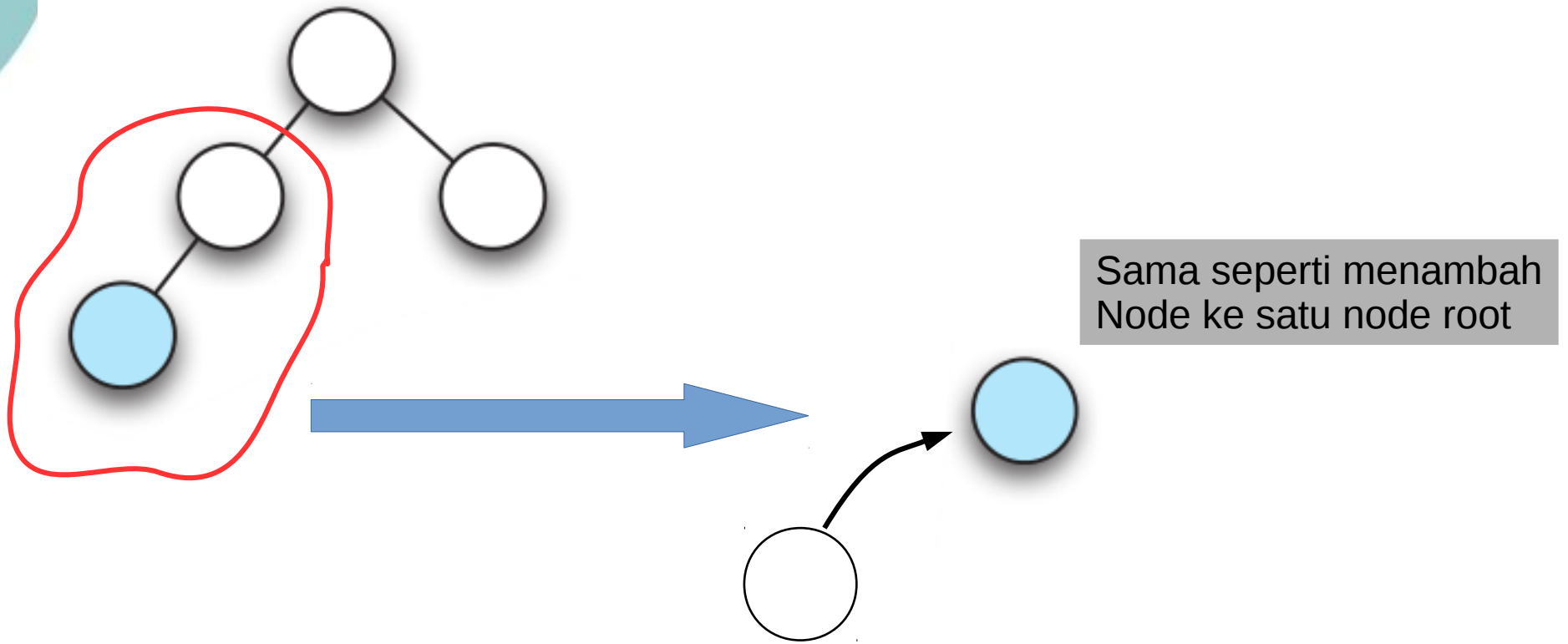


Add lagi : apakah subtree kiri lebih tinggi dari kanan ? Tidak maka insert ke sebelah kiri



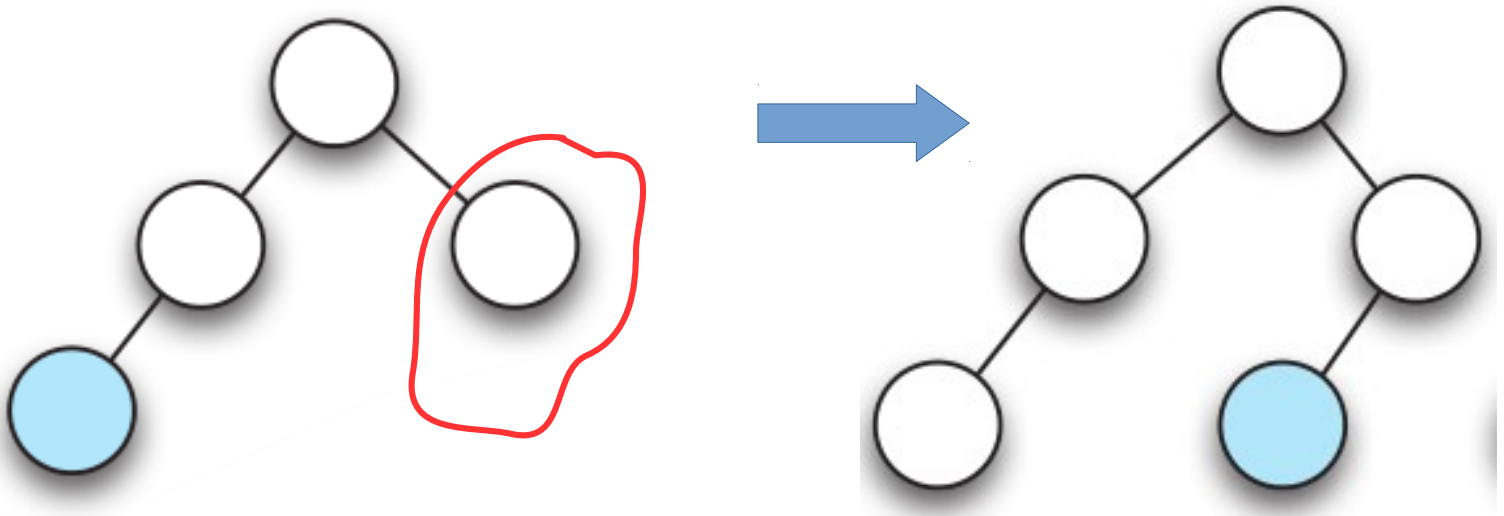
Ada proses rekursif disini

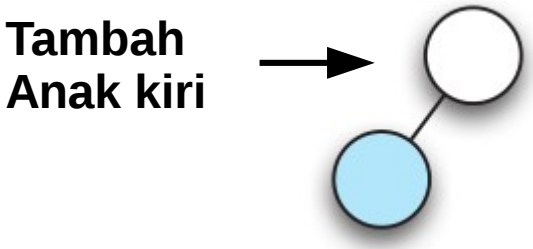
- Proses yang dilingkari merupakan pengulangan yang terjadi pada root sebelumnya



Add node (con’t)

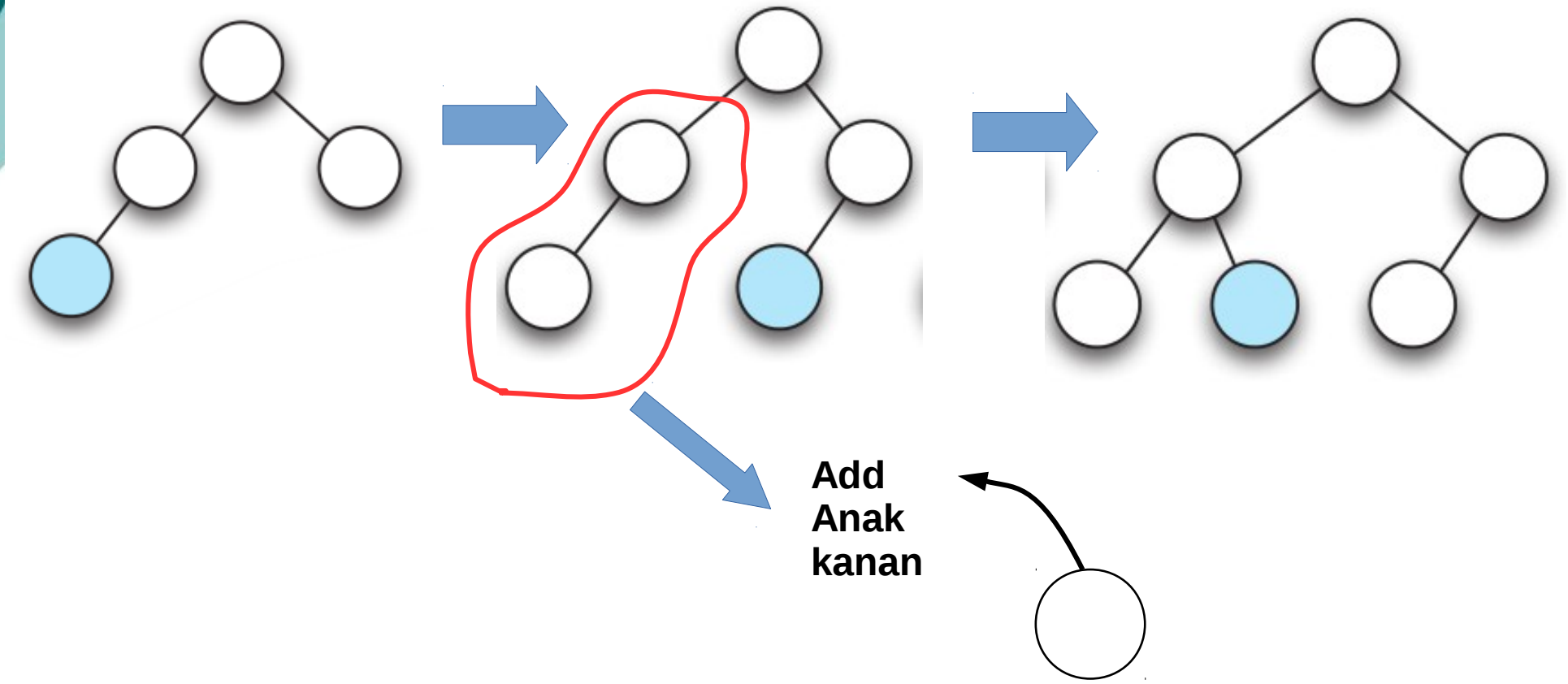
- Rekursif di subtree kanan





Add node (con’t)

- Rekursif di subtree kiri



Pseudocode Algoritma add

- Diimplementasi sebagai fungsi (mengembalikan nilai / pointer)

```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
   if subtreePtr kosong then
      return newNodePtr      //jadi root
   else
   {
      BinaryNode* anakKiri = isi anak kiri subtreePtr
      BinaryNode* anakKanan = isi anak kanan subtreePtr

      if tinggi subtree kiri > tinggi subtree kanan then
         anakKanan =
           addNode (anakKanan, newNodePtr ) // rekursif
         setAnakKanan( subtreePtr, anakKanan )
      else
         anakKiri =
           addNode (anakKiri, newNodePtr ) // rekursif
         setAnakKiri( subtreePtr, anakKiri )

      return subtreePtr
   }
```

Contoh run algoritma add

- Add ke pohon kosong :)

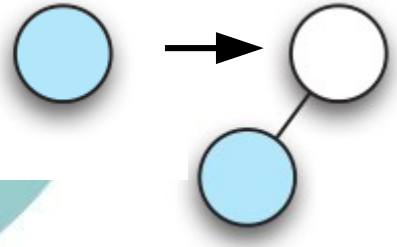
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
   if subtreePtr kosong then
      return newNodePtr      //jadi root
   else
   {
      BinaryNode* anakKiri = isi anak kiri subtreePtr
      BinaryNode* anakKanan = isi anak kanan subtreePtr

      if tinggi subtree kiri > tinggi subtree kanan then
         anakKanan =
           addNode (anakKanan, newNodePtr ) // rekursif
         setAnakKanan( subtreePtr, anakKanan )
      else
         anakKiri =
           addNode (anakKiri, newNodePtr ) // rekursif
         setAnakKiri( subtreePtr, anakKiri )

      return subtreePtr
   }
```

Run algo add (2)

- Add node ke root, root belum punya anak



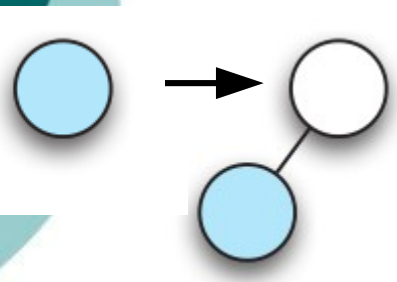
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
  if subtreePtr kosong then
     return newNodePtr      //jadi root
  else
  {
     BinaryNode* anakKiri = isi anak kiri subtreePtr
     BinaryNode* anakKanan = isi anak kanan subtreePtr

     if tinggi subtree kiri > tinggi subtree kanan then
        anakKanan =
          addNode (anakKanan, newNodePtr ) // rekursif
        setAnakKanan( subtreePtr, anakKanan )
     else
        anakKiri =
          addNode (anakKiri, newNodePtr ) // rekursif
        setAnakKiri( subtreePtr, anakKiri )

     return subtreePtr
  }
```

Run algo add (3)

- Add node ke root, root belum punya anak



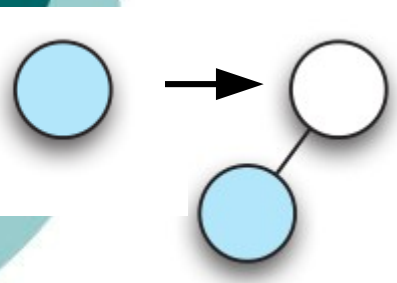
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
  if subtreePtr kosong then
     return newNodePtr      //jadi root
  else
  {
     BinaryNode* anakKiri = isi anak kiri subtreePtr
     BinaryNode* anakKanan = isi anak kanan subtreePtr

     if tinggi subtree kiri > tinggi subtree kanan then
        anakKanan =
          addNode (anakKanan, newNodePtr ) // rekursif
        setAnakKanan( subtreePtr, anakKanan )
     else
        anakKiri =
          addNode (anakKiri, newNodePtr ) // rekursif
        setAnakKiri( subtreePtr, anakKiri )

     return subtreePtr
  }
```

Run algo add (4)

- Add node ke root, root belum punya anak



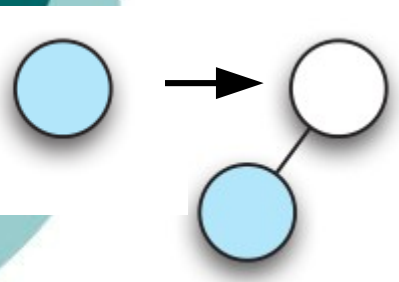
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
  if subtreePtr kosong then
     return newNodePtr      //jadi root
  else
  {
     BinaryNode* anakKiri = isi anak kiri subtreePtr
     BinaryNode* anakKanan = isi anak kanan subtreePtr

     if tinggi subtree kiri > tinggi subtree kanan then
        anakKanan =
          addNode (anakKanan, newNodePtr ) // rekursif
        setAnakKanan( subtreePtr, anakKanan )
     else
        anakKiri =
          addNode (anakKiri, newNodePtr ) // rekursif
        setAnakKiri( subtreePtr, anakKiri )

     return subtreePtr
  }
```

Run algo add (5)

- Add node ke root, root belum punya anak



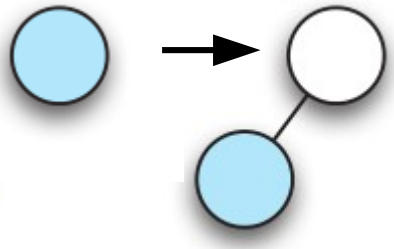
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
  if subtreePtr kosong then
     return newNodePtr      //jadi root
  else
  {
     BinaryNode* anakKiri = isi anak kiri subtreePtr
     BinaryNode* anakKanan = isi anak kanan subtreePtr

     if tinggi subtree kiri > tinggi subtree kanan then
        anakKanan =
          addNode (anakKanan, newNodePtr ) // rekursif
        setAnakKanan( subtreePtr, anakKanan )
     else
        anakKiri =
          addNode (anakKiri, newNodePtr ) // rekursif
        setAnakKiri( subtreePtr, anakKiri )

     return subtreePtr
  }
```

Run algo add (6)

- Rekursif pada subTree anak kiri dari root



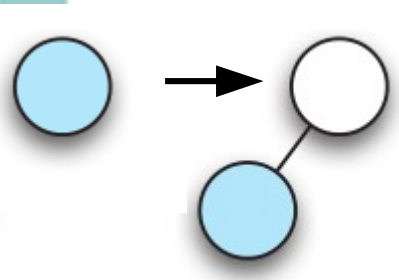
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
  if subtreePtr kosong then
     return newNodePtr      //jadi root
  else
  {
     BinaryNode* anakKiri = isi anak kiri subtreePtr
     BinaryNode* anakKanan = isi anak kanan subtreePtr

     if tinggi subtree kiri > tinggi subtree kanan then
        anakKanan =
          addNode (anakKanan, newNodePtr ) // rekursif
        setAnakKanan( subtreePtr, anakKanan )
     else
        anakKiri =
          addNode (anakKiri, newNodePtr ) // rekursif
        setAnakKiri( subtreePtr, anakKiri )

     return subtreePtr
  }
```

Run algo add (7)

- Rekursif pada subTree anak kiri dari root



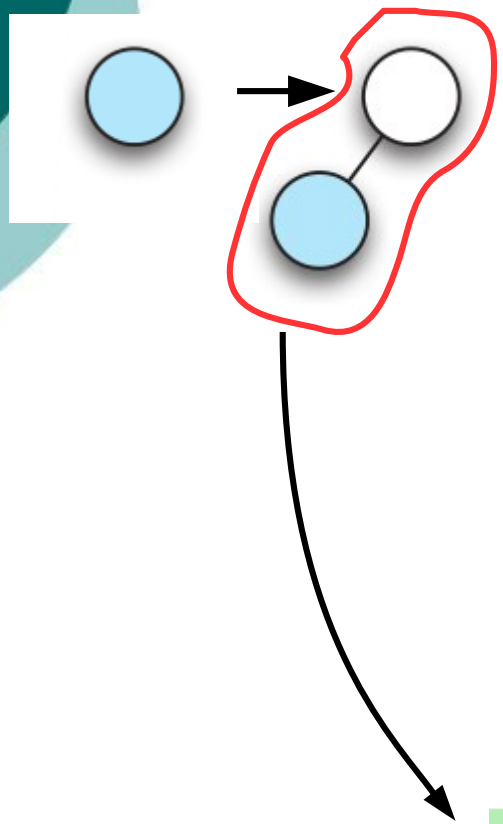
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
  if subtreePtr kosong then
     return newNodePtr      //jadi root
  else
  {
     BinaryNode* anakKiri = isi anak kiri subtreePtr
     BinaryNode* anakKanan = isi anak kanan subtreePtr

     if tinggi subtree kiri > tinggi subtree kanan then
        anakKanan =
          addNode (anakKanan, newNodePtr ) // rekursif
        setAnakKanan( subtreePtr, anakKanan )
     else
        anakKiri =
          addNode (anakKiri, newNodePtr ) // rekursif
        setAnakKiri( subtreePtr, anakKiri )

     return subtreePtr
  }
```

Run algo add (8)

- Rekursif selesai, anak kiri diupdate



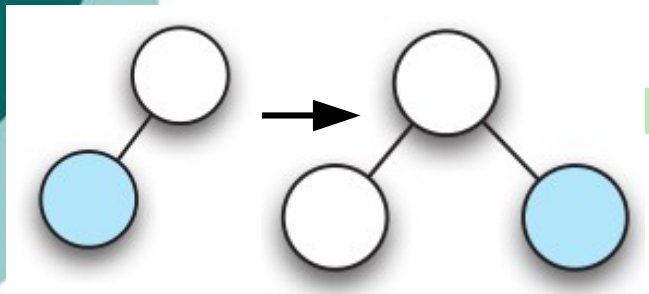
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
  if subtreePtr kosong then
     return newNodePtr      //jadi root
  else
  {
     BinaryNode* anakKiri = isi anak kiri subtreePtr
     BinaryNode* anakKanan = isi anak kanan subtreePtr

     if tinggi subtree kiri > tinggi subtree kanan then
        anakKanan =
          addNode (anakKanan, newNodePtr ) // rekursif
        setAnakKanan( subtreePtr, anakKanan )
     else
        anakKiri =
          addNode (anakKiri, newNodePtr ) // rekursif
        setAnakKiri( subtreePtr, anakKiri )

     return subtreePtr
  }
```

Run algo add (9)

- Add node ke root, root punya subtree kiri



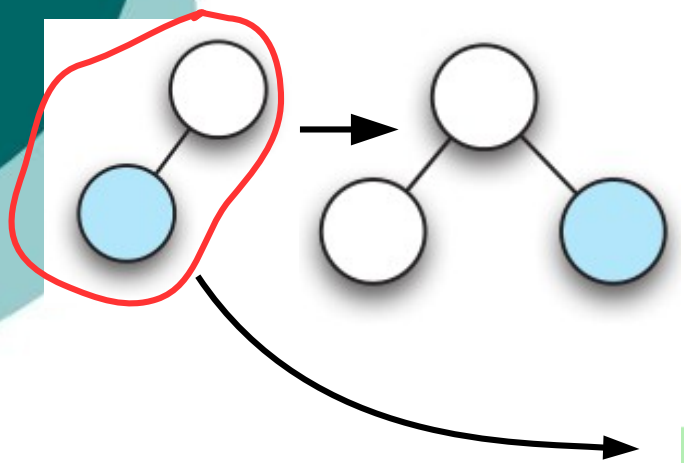
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
  if subtreePtr kosong then
     return newNodePtr      //jadi root
  else
  {
     BinaryNode* anakKiri = isi anak kiri subtreePtr
     BinaryNode* anakKanan = isi anak kanan subtreePtr

     if tinggi subtree kiri > tinggi subtree kanan then
        anakKanan =
          addNode (anakKanan, newNodePtr ) // rekursif
        setAnakKanan( subtreePtr, anakKanan )
     else
        anakKiri =
          addNode (anakKiri, newNodePtr ) // rekursif
        setAnakKiri( subtreePtr, anakKiri )

     return subtreePtr
  }
```

Run algo add (10)

- Add node ke root, root punya subtree kiri



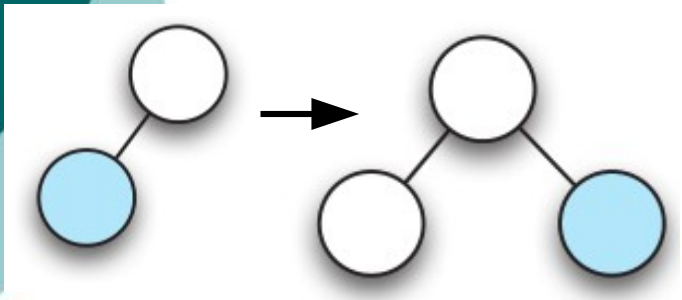
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
  if subtreePtr kosong then
     return newNodePtr      //jadi root
  else
  {
     BinaryNode* anakKiri = isi anak kiri subtreePtr
     BinaryNode* anakKanan = isi anak kanan subtreePtr

     if tinggi subtree kiri > tinggi subtree kanan then
        anakKanan =
          addNode (anakKanan, newNodePtr ) // rekursif
        setAnakKanan( subtreePtr, anakKanan )
     else
        anakKiri =
          addNode (anakKiri, newNodePtr ) // rekursif
        setAnakKiri( subtreePtr, anakKiri )

     return subtreePtr
  }
```

Run algo add (11)

- Rekursif pada subTree anak kanan dari root



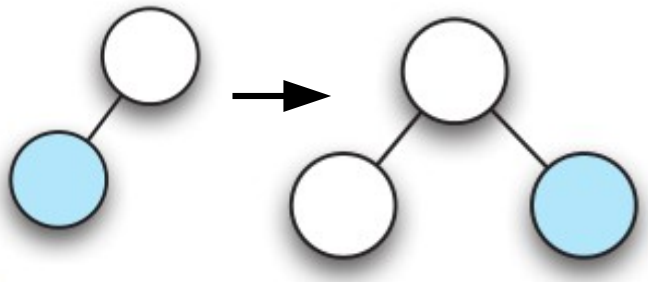
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
  if subtreePtr kosong then
     return newNodePtr      //jadi root
  else
  {
     BinaryNode* anakKiri = isi anak kiri subtreePtr
     BinaryNode* anakKanan = isi anak kanan subtreePtr

     if tinggi subtree kiri > tinggi subtree kanan then
        anakKanan =
          addNode (anakKanan, newNodePtr ) // rekursif
        setAnakKanan( subtreePtr, anakKanan )
     else
        anakKiri =
          addNode (anakKiri, newNodePtr ) // rekursif
        setAnakKiri( subtreePtr, anakKiri )

     return subtreePtr
  }
```

Run algo add (12)

- Rekursif pada subTree anak kanan dari root



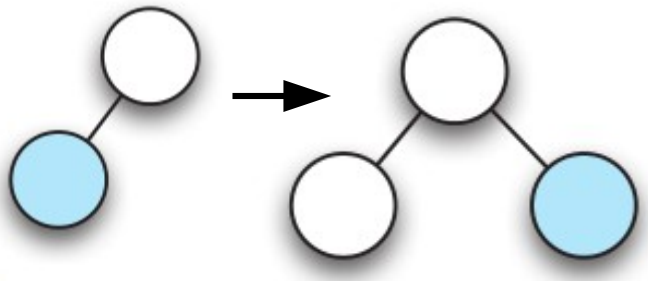
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
   if subtreePtr kosong then
      return newNodePtr      //jadi root
   else
   {
      BinaryNode* anakKiri = isi anak kiri subtreePtr
      BinaryNode* anakKanan = isi anak kanan subtreePtr

      if tinggi subtree kiri > tinggi subtree kanan then
         anakKanan =
           addNode (anakKanan, newNodePtr ) // rekursif
         setAnakKanan( subtreePtr, anakKanan )
      else
         anakKiri =
           addNode (anakKiri, newNodePtr ) // rekursif
         setAnakKiri( subtreePtr, anakKiri )

      return subtreePtr
   }
```

Run algo add (13)

- Rekursif pada subTree anak kanan dari root



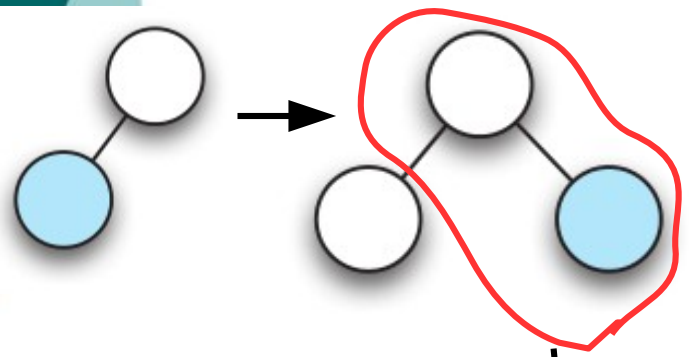
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
  if subtreePtr kosong then
     return newNodePtr      //jadi root
  else
  {
     BinaryNode* anakKiri = isi anak kiri subtreePtr
     BinaryNode* anakKanan = isi anak kanan subtreePtr

     if tinggi subtree kiri > tinggi subtree kanan then
        anakKanan =
          addNode (anakKanan, newNodePtr ) // rekursif
        setAnakKanan( subtreePtr, anakKanan )
     else
        anakKiri =
          addNode (anakKiri, newNodePtr ) // rekursif
        setAnakKiri( subtreePtr, anakKiri )

     return subtreePtr
  }
```

Run algo add (14)

- Rekursif selesai, anak kanan diupdate



Catat Pointer ke Node biru Sbg anak kanan

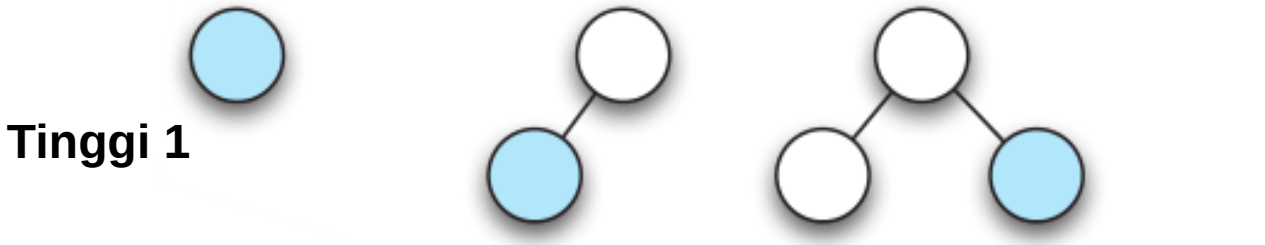
```
function addNode( BinaryNode* subtreePtr,
                  BinaryNode* newNodePtr )
  if subtreePtr kosong then
     return newNodePtr      //jadi root
  else
  {
     BinaryNode* anakKiri = isi anak kiri subtreePtr
     BinaryNode* anakKanan = isi anak kanan subtreePtr

     if tinggi subtree kiri > tinggi subtree kanan then
        anakKanan =
          addNode (anakKanan, newNodePtr ) // rekursif
        setAnakKanan( subtreePtr, anakKanan )
     else
        anakKiri =
          addNode (anakKiri, newNodePtr ) // rekursif
        setAnakKiri( subtreePtr, anakKiri )

     return subtreePtr
  }
```

Bagaimana mengukur tinggi subtree?

- Fungsi mengembalikan tinggi pohon dihitung secara rekursif
- HitungTinggi(subTree) :
  - Jika subTree kosong maka tinggi = 0
  - Jika subtree tidak kosong: 1 + max(HitungTinggi(Subtree kiri), Hitung Tinggi (Subtree kanan))



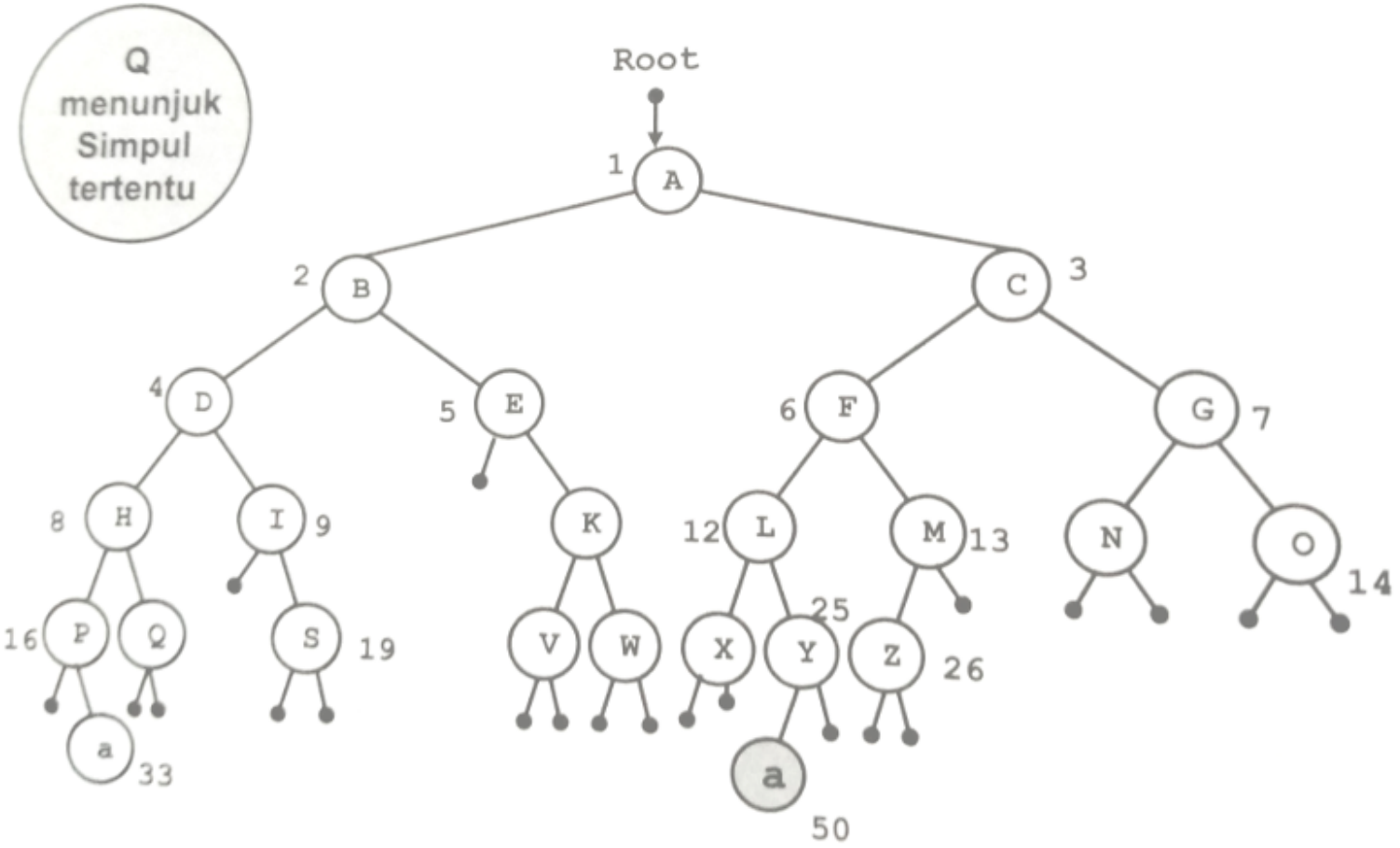
Insert simpul pada posisi tertentu dalam pohon

- Pre-kondisi :

  sudah ada pohon (minimal ada root), bukan pohon kosong
- Kita ingin menyisipkan node sesuai nomer urut nya (lihat halaman penomoran)
- Maka algoritma / program perlu mencari dahulu posisi nomer yang akan disisipkan

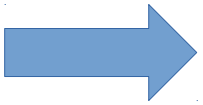
Contoh

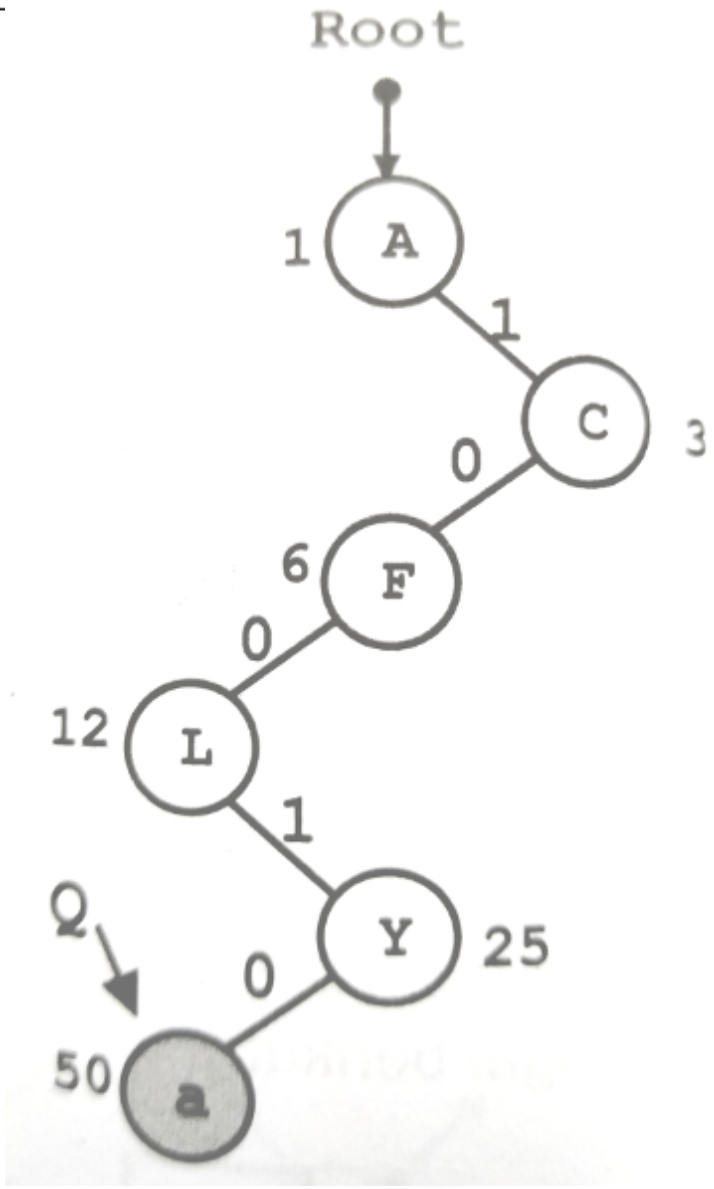
- Kita ingin menyisipkan node **a (nomor 50)** yang nantinya ditunjuk oleh pointer Q



Cara mencari posisi

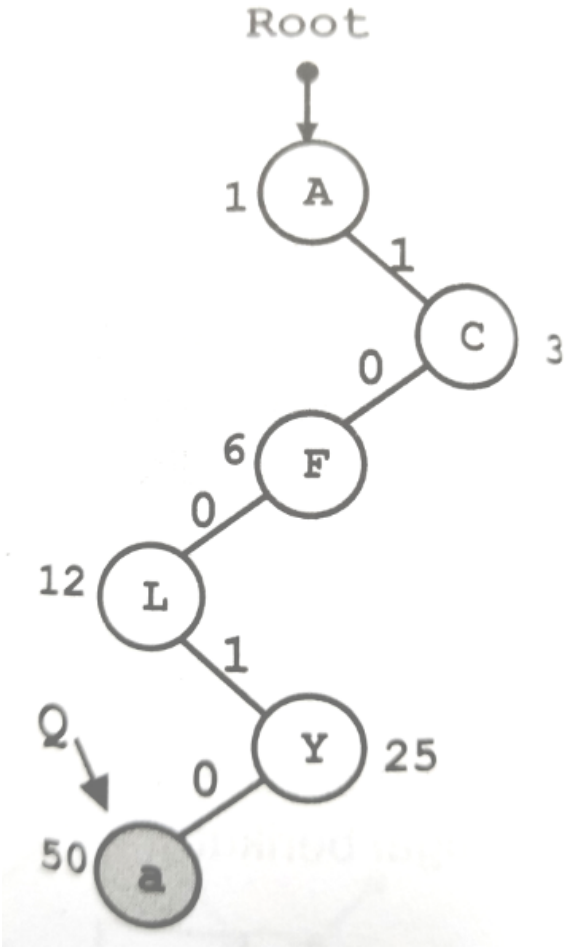
```
Q = Root;     //akar
Q = Q->right;//no.3
Q = Q->left; //no.6
Q = Q->left; //no.12
Q = Q->right;//no.25
Q = Q->left; //no.50
```





Algoritma penelusuran

- Bagaimana agar algoritma penelusuran bersifat general (umum) dan fleksibel ?



Pergerakan penelusuran bisa diringkas
Dalam bentuk angka : jika 1 ke kanan
Jika 0 ke kiri

1	0	0	1	0

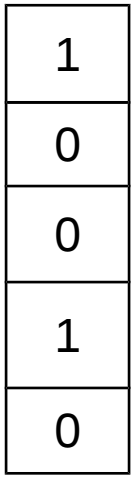
1
0
0
1
0

Jika kita taruh dalam stack
Elemen ke-0 pada array diatas
Akan ditempatkan di top

Menelusuri dengan stack

- Algoritma menelusuri posisi node dengan menaruh angka 0 dan 1 ke dalam stack
- Caranya : input = 50 (nomor node yang akan disisipkan)

n=50	Sisa
50 / 2 = 25	0
25 / 2 = 12	1
12 / 2 = 6	0
6 / 2 = 3	0
3 / 2 = 1	1



Algoritma

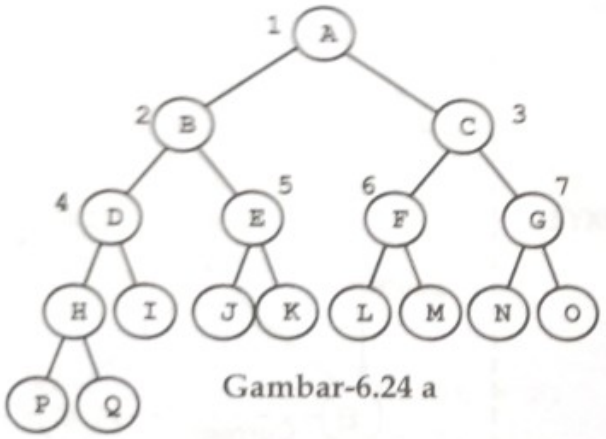
```
Stack s;
n = 50;    //nomor node yg akan disisip
hasil = n;
while (hasil > 1)  //looping selama hasil >1
  {
    sisa = hasil % 2; //operator bagi sisa
    s.push( sisa );
    hasil = hasil /2; //update hasil
  }
```

Membaca pohon biner

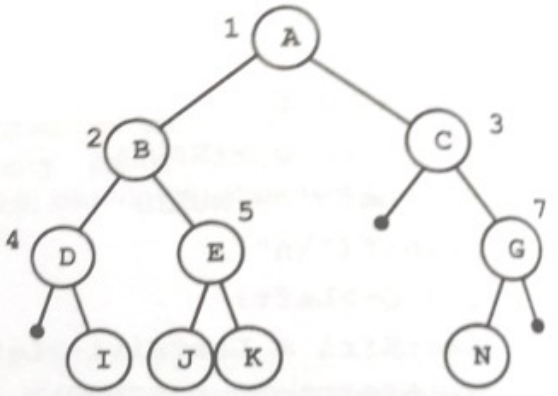
- Pre-kondisi: pohon sudah ada di memori (sudah disusun)
- Ada dua cara membaca pohon biner :
  - Membaca level per level (semua node dibaca)
  - Membaca atau mencari berdasar nomor node (hanya node tertentu)
- Setelah data terbaca, kita bisa mencetak ke layar atau proses lanjutan lainnya

Contoh baca pohon

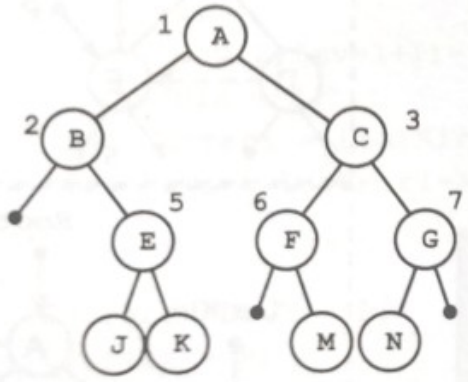
- Diberikan contoh pohon berikut:



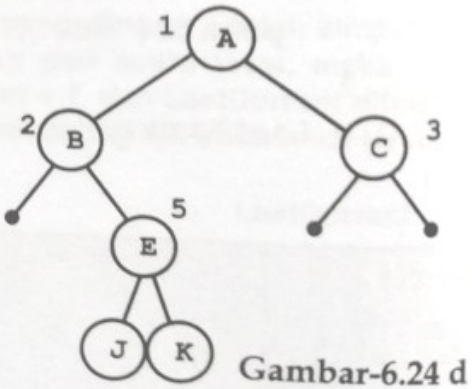
Gambar-6.24 a



Gambar-6.24 b



Gambar-6.24 c



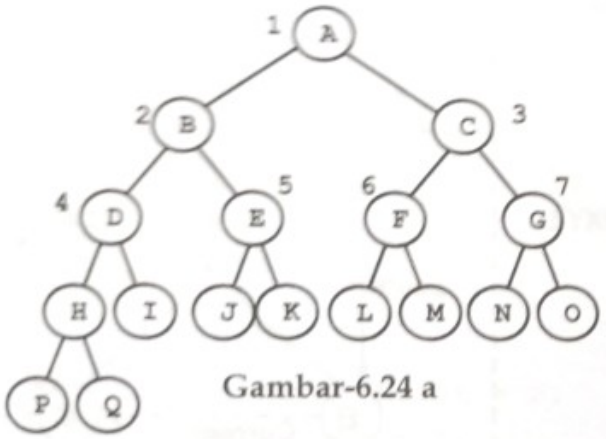
Gambar-6.24 d

Pohon gambar a dibaca:

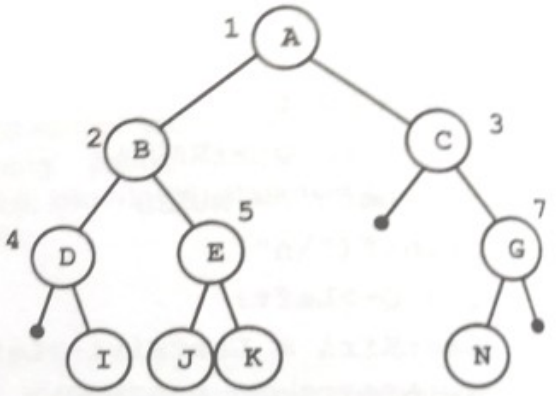
A B C D E F G H I J K L M N O P Q

Contoh baca pohon

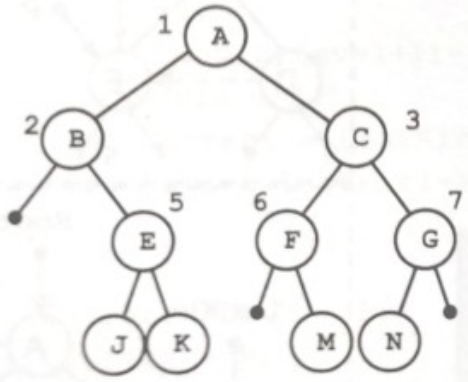
- Diberikan contoh pohon berikut:



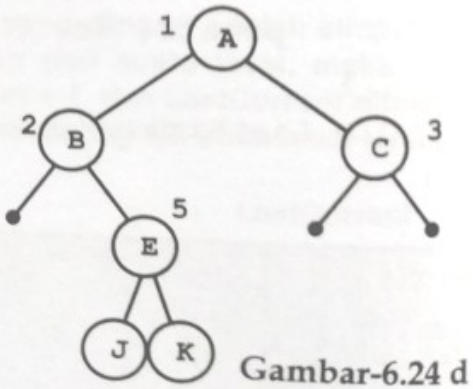
Gambar-6.24 a



Gambar-6.24 b



Gambar-6.24 c



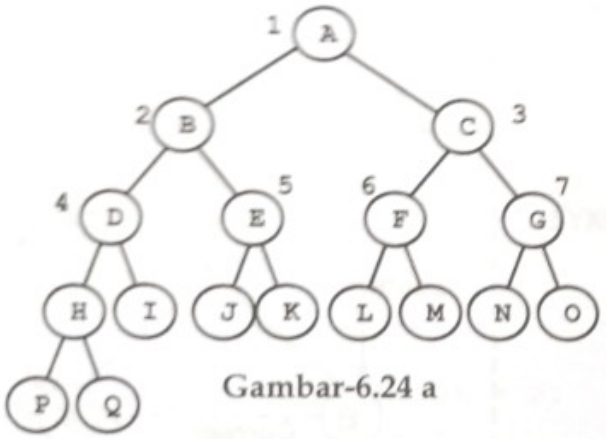
Gambar-6.24 d

Pohon gambar b dibaca:

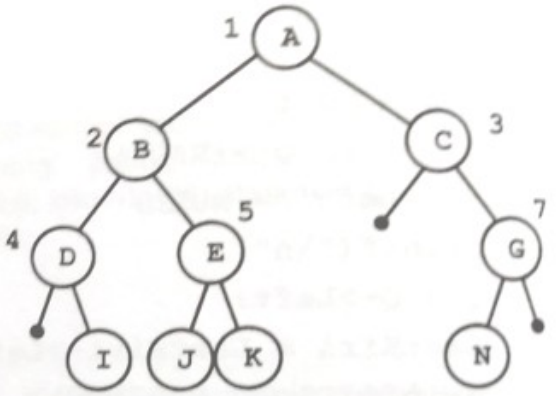
A B C D E G I J K N

Contoh baca pohon

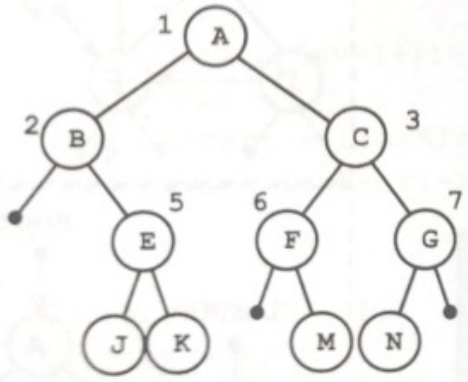
- Diberikan contoh pohon berikut:



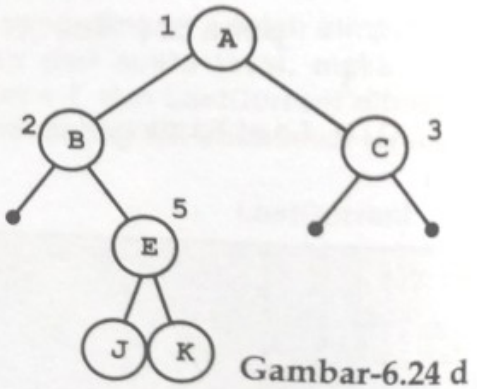
Gambar-6.24 a



Gambar-6.24 b



Gambar-6.24 c



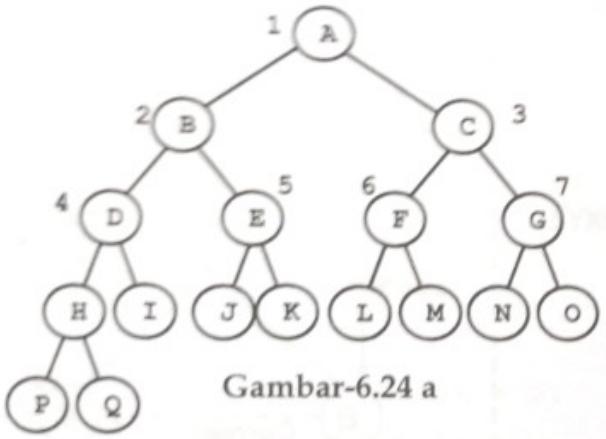
Gambar-6.24 d

Pohon gambar c dibaca:

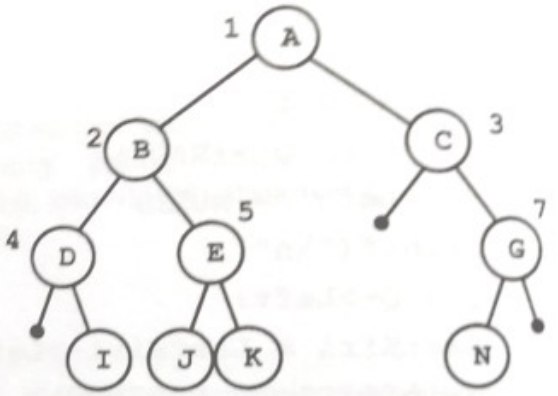
A B C E F G J K M N

Contoh baca pohon

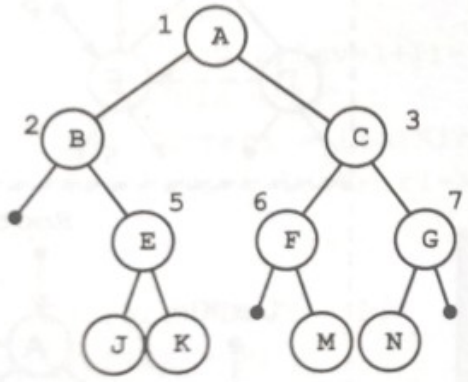
- Diberikan contoh pohon berikut:



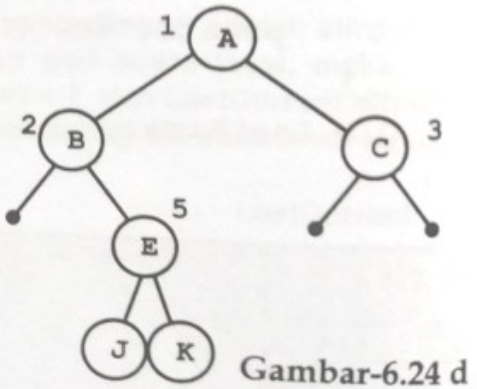
Gambar-6.24 a



Gambar-6.24 b



Gambar-6.24 c



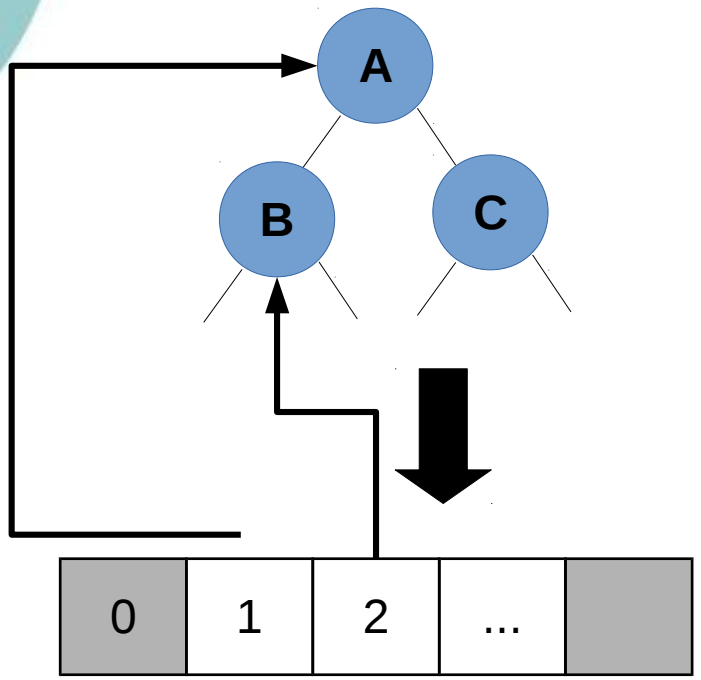
Gambar-6.24 d

Pohon gambar d dibaca:

A B C E J K

Baca pohon

- Proses baca : data dari memori di pindah ke representasi array (tabel)



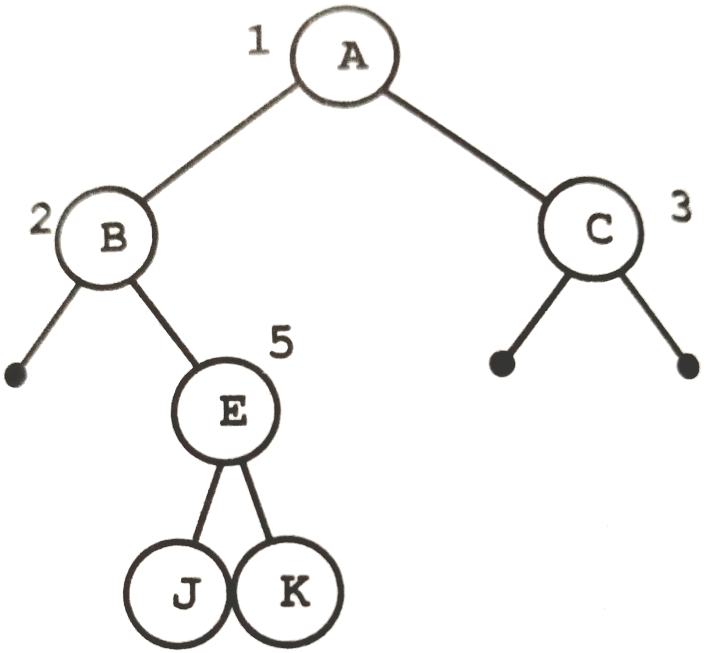
Q : array of pointer to Node

```
NodeTree *Root,*Current;
Q[1] = Root;  //elemen ke-1 diisi root
i = 1; j = 1; //pencacah (mencatat posisi)
//memasukkan alamat node di memori ke array
while (Q[i] != NULL) //pointer Tidak kosong
  {
    Current = Q[i];
    print_Node( Current->info ); //cetak info
    if (Current->left != NULL)//ada anak kiri
      {
        j++;
        Q[j]=Current->left;//simpan ke array Q
      }
    if (Current->right != NULL)//ada anak kanan
      {
        j++;
        Q[j]=Current->right;//simpan ke array Q
      }
    i++;
  }
```

Contoh hasil baca pohon

- Diberikan pohon berikut maka Loop pertama (i=1; j=1) dihasilkan

Q[1] = Root;



0	&A	0	...	

Ada anak kiri

```
{
   j++;
  Q[j] = Current→left;
}
```

i=1; j=2 Current=A

i=1 j=2

0	&A	&B	...	

Contoh hasil baca (2)

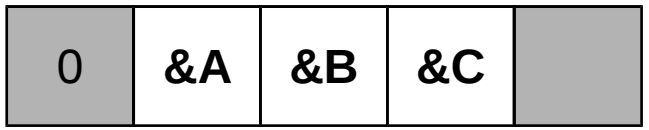
- Kondisi keluar dari loop pertama

Ada anak kanan

```
{
   j++;
  Q[j] = Current→right;
}
```

i=1; j=3 Current=A

i=1 j=3

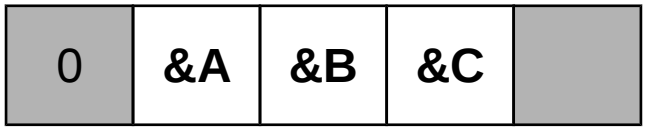


Perintah di bagian akhir Loop while

```
i++;
```

i=2; j=3 Current=A

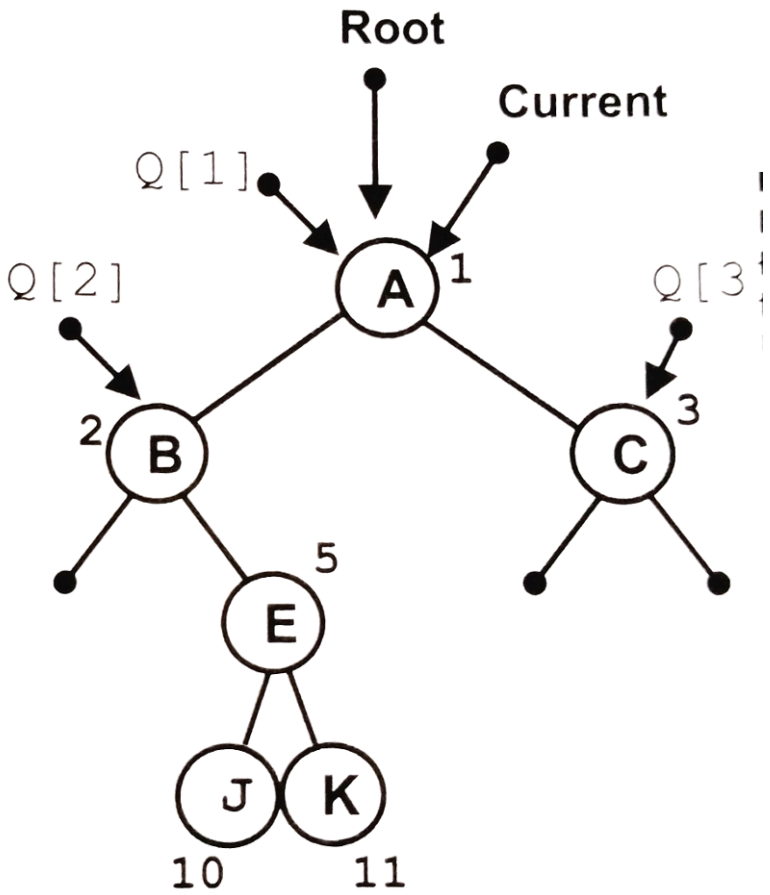
i=2 j=3



Contoh hasil baca (3)

- Masuk loop kedua (i=2; j=3)

Current = Q[i]; // Node B



Ada anak kanan

```
{
  j++;
  Q[j] = Current→right;
}
```

i=2; j=4 Current=B

i=2 j=4

0	&A	&B	&C	&E

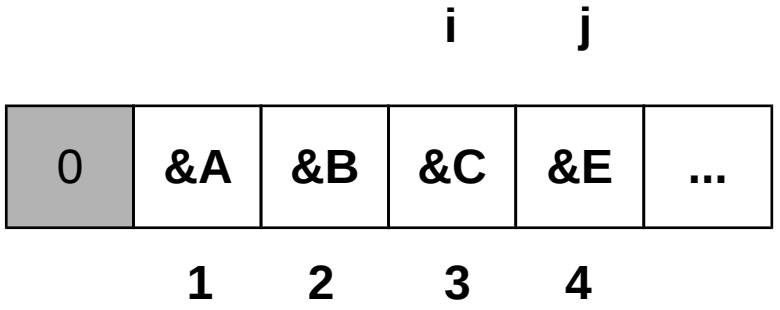
Tidak ada anak kiri maka lewati

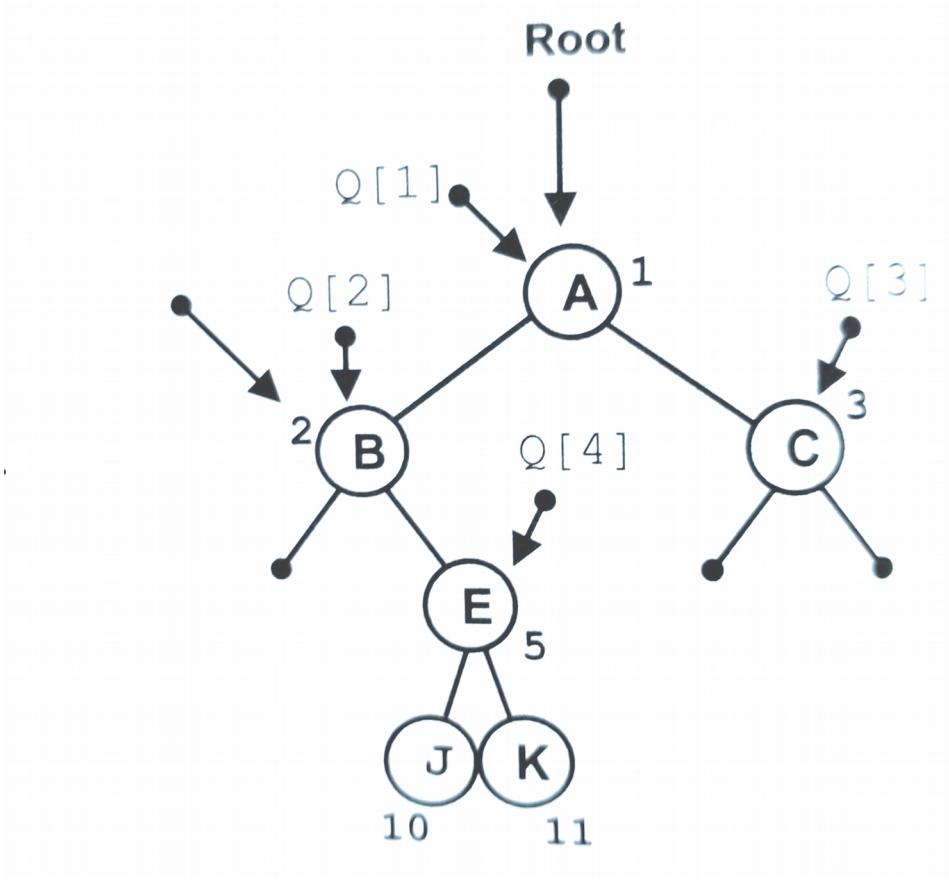
Contoh hasil baca (3)

- Keluar dari loop kedua

i++; // i jadi 3

i=3; j=4 Current = B






Thank
You
Obrigado
Merci
Asante
Multumesc
Spasibo
Maake
Kia Ora
Ua Tsaug Rau Koj
Matur Nuwun
Asante
Kiitos
Salamat
Obrigado
Ua Tsaug Rau Koj
Kiitos
Chokrane
Mochchakkeram
Spasibo
Maake
Chokrane
Ua Tsaug Rau Koj
Matur Nuwun
Grazie
Asante
Obrigado
Kiitos
Welalin
Multumesc
Chokrane
Chokrane
Matondo
Chokrane
Kiitos
Chokrane
Dankon
Mochchakkeram
Mochchakkeram
Obrigado
Matondo
Asante
Ua Tsaug Rau Koj
Mochchakkeram
Obrigado
Grazie
Mochchakkeram
Grazie
Mochchakkeram
Mamana
Grazie
Ua Tsaug Rau Koj
Nirringrazzjak
Juspaxar
Spasibo
Raibh Maith Agat
Mochchakkeram
Nirringrazzjak
Dank Je
Arigato
Mochchakkeram
Merci
Spasibo
Obrigado
Raibh Maith Agat
Mochchakkeram
Ua Tsaug Rau Koj
Ua Tsaug Rau Koj
Matondo
Mochchakkeram
Vinaka
Cam on ban
Asante
Grazie
Salamat
Kiitos
Chokrane
Grazie
Terma Kasih
Ua Tsaug Rau Koj
Multumesc
Welalin
Grazie
Mochchakkeram
Merci
Kia Ora
Obrigado
Chokrane