Pohon Biner (Bagian 1)

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Tujuan

Mahasiswa memahami definisi pohon dan pohon biner

Berdasarkan pemahaman tersebut, mampu membuat fungsi sederhana yang memanipulasi pohon

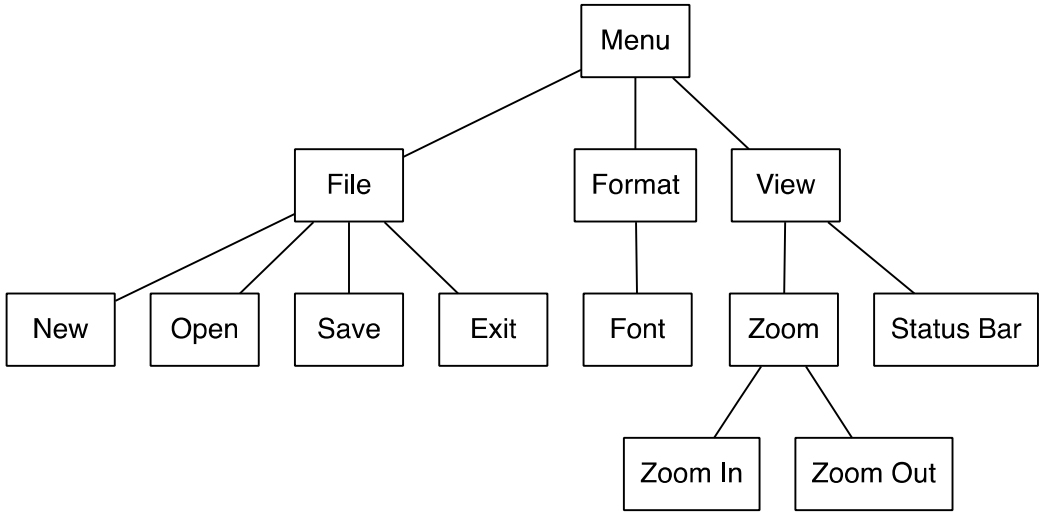
Mahasiswa mampu mengimplementasi fungsi pemroses pohon dalam bahasa C (melalui praktikum)

Contoh Persoalan - 1

Menu dalam Aplikasi Komputer

Contoh (Notepad):

- File
  - New
  - Open
  - Save
  - Exit
- Format
  - Font
- View
  - Zoom
    - Zoom In
    - Zoom Out
  - Status Bar



Contoh Persoalan - 2

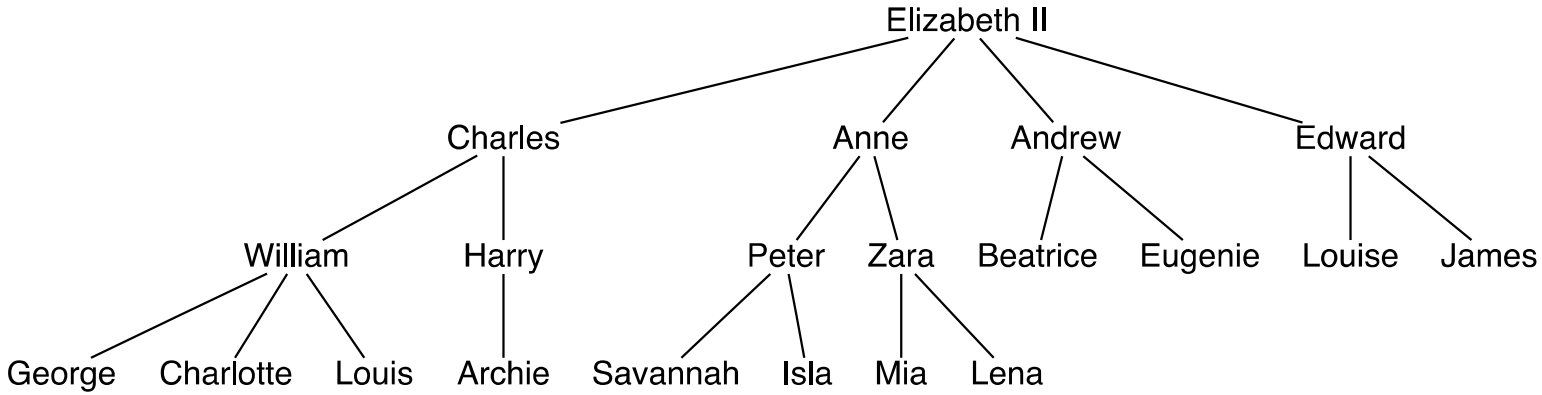
Susunan bab dalam buku
Contoh: Diktat Struktur Data

- Bagian I. Struktur Data
  - Abstract Data Type
    - ADT JAM dalam Bahasa Algoritmik
    - ADT POINT dalam Bahasa Algoritmik
    - ADT GARIS dalam Bahasa Algoritmik
    - Latihan Soal
  - Koleksi Objek
  - …
- Bagian II
  - Studi Kasus 1 Polinom
    - Deskripsi Persoalan
    - …
  - Studi Kasus 2 Kemunculan Huruf dan Posisi Pada Pita Karakter
    - Deskripsi Persoalan
    - …
  - …

Contoh Persoalan - 3

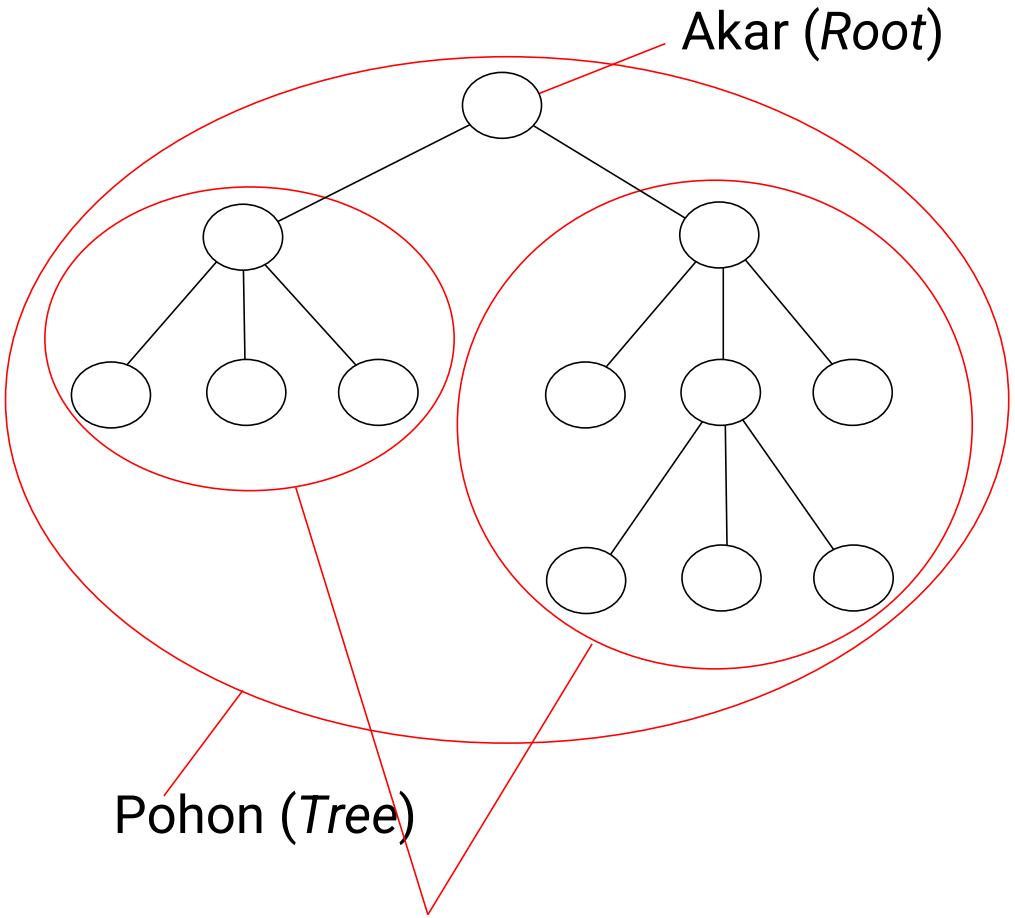
Pohon keluarga

Contoh: Pohon keluarga bangsawan Inggris



Definisi

Pohon



Akar

SubPohon

Definisi Rekursif Pohon:
- Akar → basis
- Sub Pohon (sub himpunan yang berupa pohon) → rekurens

Definisi Rekursif Pohon

Pohon (tree) adalah himpunan terbatas, tidak kosong, dengan elemen sebagai berikut:

- Sebuah elemen yang dibedakan dari yang lain → AKAR
- Elemen yang lain (jika ada) dibagi-bagi menjadi beberapa sub himpunan yang disjoin dan masing-masing sub himpunan itu adalah pohon → SUBPOHON

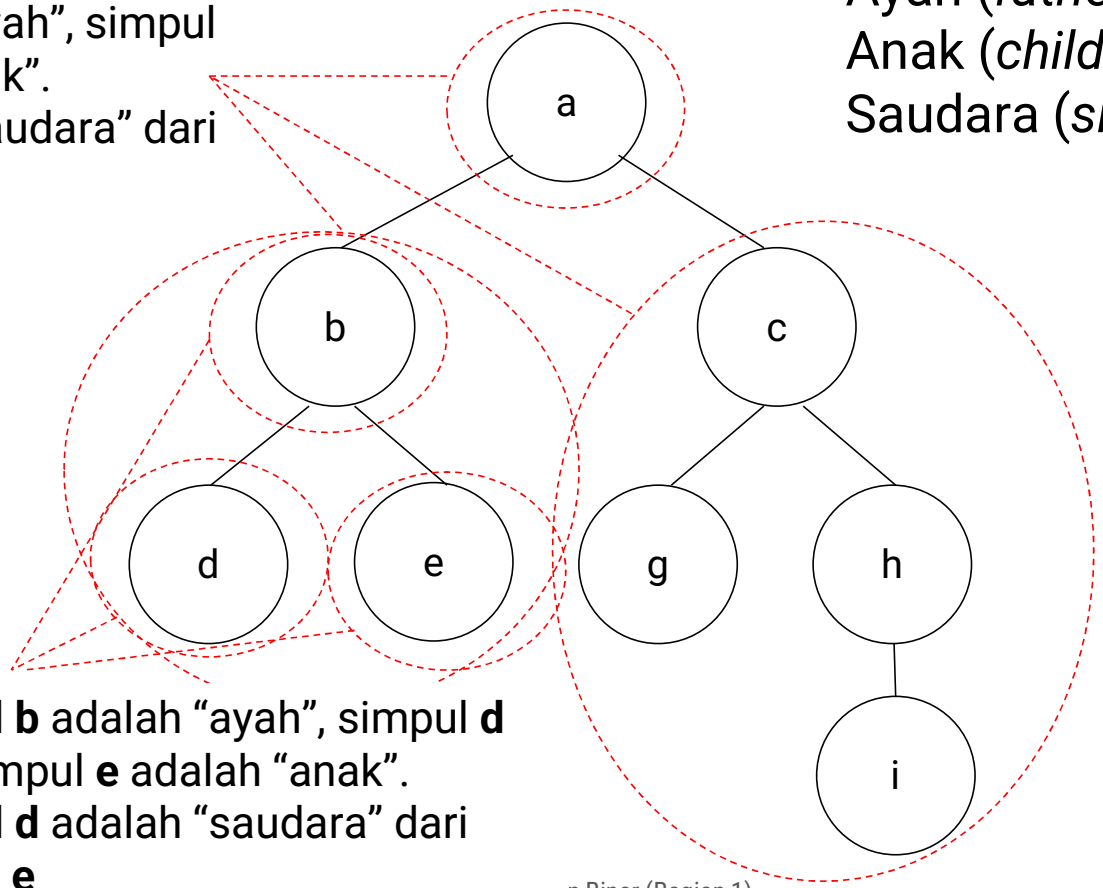
Suffiks -aire pada pohon menunjukkan berapa maksimum subpohon yang dapat dimiliki oleh suatu pohon

- Binaire (binary), maksimum subpohon: 2
- N-aire, maksimum subpohon: N

Istilah

Simpul **a** adalah “ayah”, simpul **b** dan simpul **c** “anak”.
Simpul **b** adalah “saudara” dari simpul **c**

Ayah (*father/parent*)
Anak (*child*)
Saudara (*sibling*)



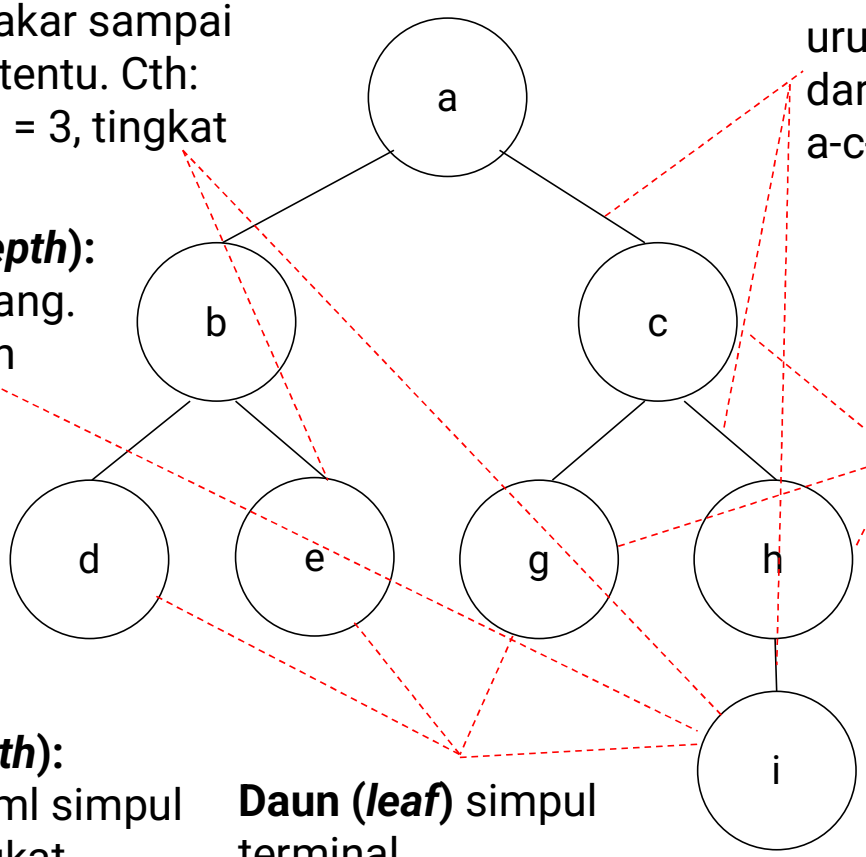
Simpul **b** adalah “ayah”, simpul **d** dan simpul **e** adalah “anak”.
Simpul **d** adalah “saudara” dari simpul **e**

Istilah

Tingkat (*level*): panjang jalan dari akar sampai simpul tertentu. Cth: tingkat (e) = 3, tingkat (i) = 4,

Kedalaman (*depth*): tingkat terpanjang. Cth: kedalaman pohon=4

Jalan (*path*): urutan tertentu dari cabang, cth: a-c-h-i

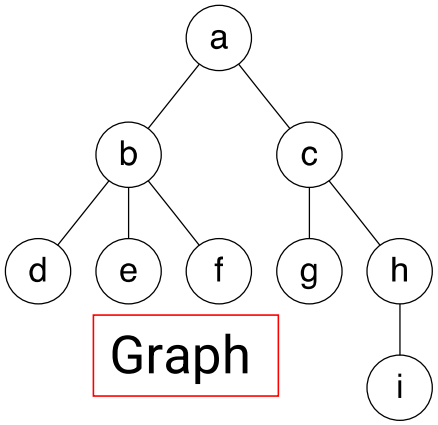


Derajat (*degree*): banyaknya anak sebuah simpul. Cth, derajat(c)=2, derajat(h)=1, derajat(g)=0

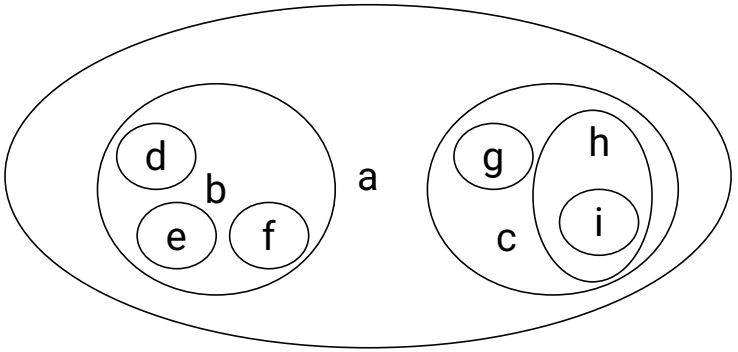
Lebar (*breadth*): maksimum jml simpul pd suatu tingkat.

Daun (*leaf*) simpul terminal

Beberapa Ilustrasi Representasi



Graph



Himpunan

Indentasi

```
a
   b
      d
      e
      f
   c
      g
      h
         i
```

Bentuk Linier

Prefix:
- (a (b (d (), e (), f ()), c (g (), h (i ()))))
- (a (b (d) (e) (f)) (c (g) (h (i))))

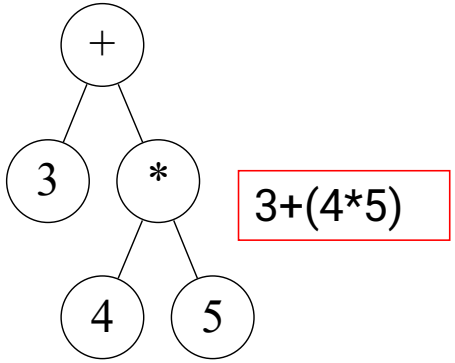
Postfix: (((d, e, f) b, (g, (i) h) c) a)

Pohon Biner

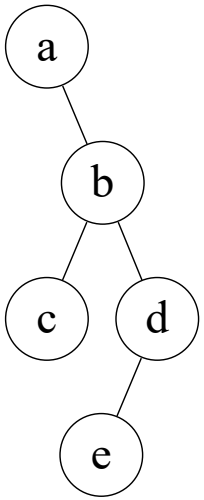
pohon biner adalah himpunan terbatas yang

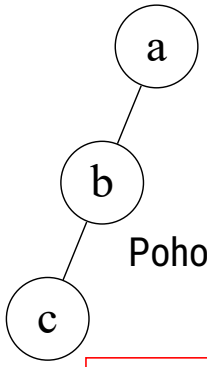
- mungkin **kosong**, atau
- terdiri atas sebuah simpul yang disebut **akar** dan dua buah himpunan lain yang *disjoint* yang merupakan pohon biner, yang disebut sebagai **sub pohon kiri** dan **sub pohon kanan** dari pohon biner tersebut

Contoh Pohon Biner

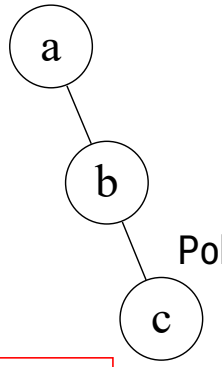


3+(4*5)





Pohon biner condong kiri



Pohon biner condong kanan

Pohon condong/skewed tree

ADT Pohon Biner dengan Representasi Berkait

```
KAMUS
{ Deklarasi TYPE  POHON BINER }
    constant  NIL: ... { konstanta pohon kosong, terdefinisi }

    type   ElType: ... { terdefinisi }
    type  Address: ... { terdefinisi }
    { Type Pohon Biner }
    type  BinTree: Address
    type TreeNode: < info:  ElType,   { simpul/akar }
                     left:  BinTree,  { subpohon kiri }
                    right:  BinTree   { subpohon kanan } >

{ Tambahan struktur data list untuk pengelolaan elemen pohon }
    type Node: < info: ElType,
                 next: AddressList >
    type NodeList: AddressList
    { list linier yang elemennya adalah Node }
```

Struktur Data Pohon Biner untuk Pemrosesan secara Rekursif (Bahasa C, pointer)

```
#define NIL NULL

/* Selektor */
#define  ROOT(p) (p)->info
#define  p↑.left (p)->left
#define p↑.right (p)->right

typedef int ElType;
typedef struct treeNode* Address;
typedef struct treeNode {
     ElType info;
     Address left;
     Address right;
} TreeNode;
/* Definisi PohonBiner */
/* pohon Biner kosong p = NIL */

typedef Address BinTree;
```

Konstruktor

```
function NewTree (akar: ElType, l: BinTree, r: BinTree) → BinTree
{ Menghasilkan sebuah pohon biner dari akar, l, dan r, jika alokasi
  berhasil }
{ Menghasilkan pohon kosong (NIL) jika alokasi gagal }

procedure CreateTree (input akar: ElType,
                      input l: BinTree,  input r: BinTree,
                     output p: BinTree)
{ I.S. Sembarang }
{ F.S. Menghasilkan sebuah pohon p }
{ Menghasilkan sebuah pohon biner p dari akar, l, dan r, jika alokasi
  berhasil }
{ Menghasilkan pohon p yang kosong (NIL) jika alokasi gagal }
```

Selektor

Jika ***p*** adalah **BinTree**, maka:

Akar dari *p* adalah ***p*↑.info**

Anak kiri *p* atau **subpohon kiri *p*** adalah ***p*↑.left**

Anak kanan *p* atau **subpohon kanan *p*** adalah ***p*↑.right**

Memory Management

```
function newTreeNode (x: ElType) → Address
{ Mengirimkan address hasil alokasi sebuah elemen bernilai x }
{ Jika alokasi berhasil, maka address tidak NIL, dan misalnya
  menghasilkan p, maka p↑.info=x, p↑.left=NIL, p↑.right=NIL }
{ Jika alokasi gagal, mengirimkan NIL }

procedure deallocTreeNode (input/output p: Address)
{ I.S. p terdefinisi }
{ F.S. p dikembalikan ke sistem }
{ Melakukan dealokasi/pengembalian address p }
```

Catatan: untuk NodeList harus dibuat primitif memory management sendiri

Predikat Penting - 1

```
function isTreeEmpty (p: BinTree) → boolean
{ Mengirimkan true jika p adalah pohon biner yang kosong }
KAMUS LOKAL
  -
ALGORITMA
  → (p = NIL)
```

```
function isOneElmt (p: BinTree) → boolean
{ Mengirimkan true jika p tidak kosong dan hanya terdiri atas 1 elemen }
KAMUS LOKAL
  -
ALGORITMA
  if not(isTreeEmpty(p))
    → ((p↑.left = NIL) and (p↑.right = NIL))
  else
    → false
```

h1

h1 versi sebelumnya:
--> not(IsTreeEmpty(L)) and
(Left(P) = Nil) and (Right(P) = Nil)
hp; 21/11/2010

Predikat Penting - 2

```
function isUnerLeft (p: BinTree) → boolean
{ Mengirimkan true jika pohon biner tidak kosong, p adalah pohon unerleft:
  hanya mempunyai subpohon kiri }

function isUnerRight (p: BinTree) → boolean
{ Mengirimkan true jika pohon biner tidak kosong, p adalah pohon unerright:
  hanya mempunyai subpohon kanan }

function isBiner (p: BinTree) → boolean
{ Mengirimkan true jika pohon biner tidak kosong, p adalah pohon biner:
  mempunyai subpohon kiri dan subpohon kanan }
```

Pohon Basis-0

Definisi rekursif

Basis: pohon biner kosong adalah pohon biner {menggunakan predikat `isTreeEmpty`}

Rekurens: Pohon biner tidak kosong terdiri dari sebuah simpul akar dan dua anak: (i) sub pohon kiri dan (ii) sub pohon kanan. Sub pohon kiri dan sub pohon kanan adalah pohon biner

Pohon Basis-1

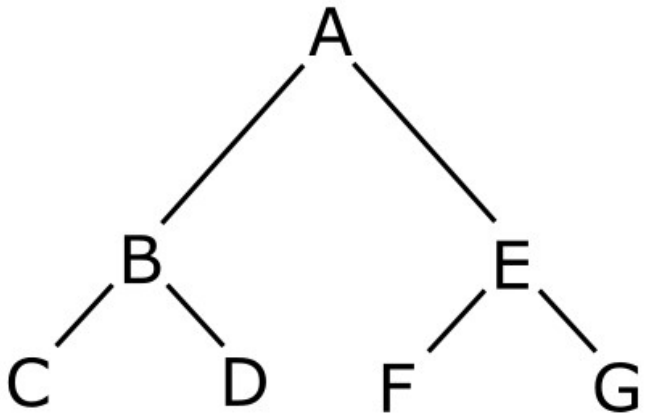
Definisi rekursif

Basis: pohon biner yang hanya terdiri dari akar {menggunakan predikat **`isOneElmt`**}

Rekurens: Pohon biner tidak kosong terdiri dari sebuah simpul akar dan dua anak yang **salah satunya pasti tidak kosong**: sub pohon kiri dan sub pohon kanan.

Gunakan **`isUnerLeft`**, **`isUnerRight`**, **`isBiner`** untuk memastikan tidak terjadi pemrosesan pada pohon kosong

Pemrosesan Traversal



pre-order: pemrosesan dengan urutan "**akar – kiri – kanan**"

- Urutan pemrosesan:
  A-B-C-D-E-F-G

in-order: pemrosesan dengan urutan "**kiri – akar– kanan**"

- Urutan pemrosesan:
  C-B-D-A-F-E-G

post-order: pemrosesan dengan urutan "**kiri – kanan – akar**"

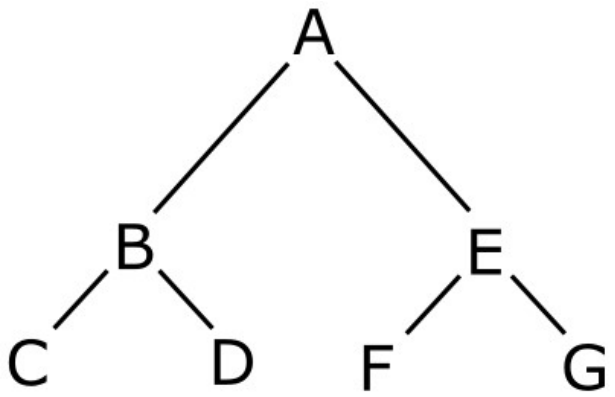
- Urutan pemrosesan:
  C-D-B-F-G-E-A

Slide 22

AU1 Saya tambahkan overview high levelnya dulu sebelum masuk ke masing2 pemrosesan traversal
Ardian Umam; 02/11/2019

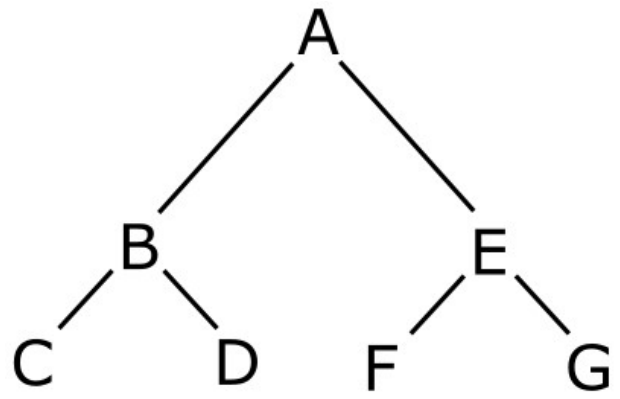
Traversal - Preorder

```
procedure preOrder (input p: BinTree)
{ I.S. Pohon p terdefinisi }
{ F.S. Semua node pohon p sudah diproses secara pre-order:
       akar, kiri, kanan }
{ Basis Pohon kosong tidak ada yang diproses }
{ Rekurens Proses akar p;
           Proses subpohon kiri p secara pre-order
           Proses subpohon kanan p secara pre-order }
KAMUS LOKAL
  -
ALGORITMA
  if isTreeEmpty(p) then { Basis-0 }
    { do nothing }
  else { Rekurens, tidak kosong }
    proses(p)
    preOrder(p↑.left)
    preOrder(p↑.right)
```



Contoh - PrintPreOrder

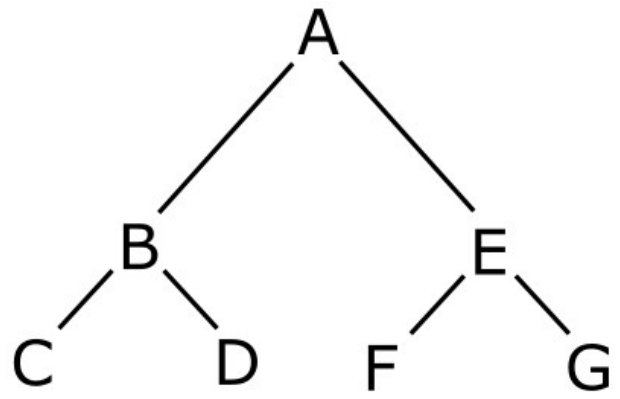
```
procedure printPreOrder (input p: BinTree)
{ I.S. Pohon p terdefinisi }
{ F.S. Semua node pohon p sudah dicetak secara pre-order:
       akar, kiri, kanan }
{ Basis Pohon kosong tidak ada yang diproses }
{ Rekurens Cetak akar p;
           Cetak subpohon kiri p secara pre-order
           Cetak subpohon kanan p secara pre-order }
KAMUS LOKAL
  -
ALGORITMA
  if isTreeEmpty(p) then { Basis-0 }
    { do nothing }
  else { Rekurens, tidak kosong }
    output(p↑.info)
    printPreOrder(p↑.left)
    printPreOrder(p↑.right)
```



Urutan output = A-B-C-D-E-F-G

Traversal - Inorder

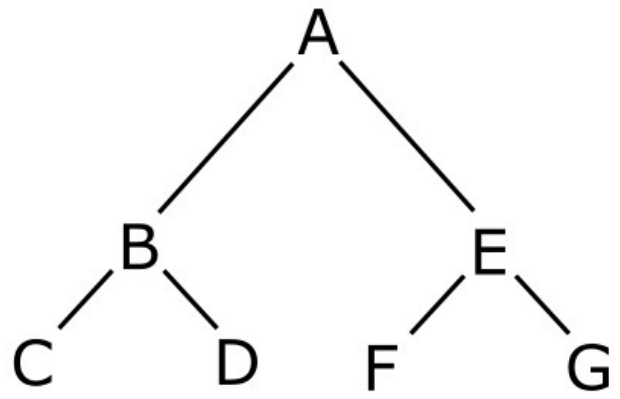
```
procedure inOrder (input p: BinTree)
{ I.S. Pohon p terdefinisi }
{ F.S. Semua node pohon p sudah diproses secara InOrder:
       kiri, akar, kanan }
{ Basis Pohon kosong tidak ada yang diproses }
{ Rekurens Proses subpohon kiri p secara in-order
           Proses akar p;
           Proses subpohon kanan p secara in-order }
KAMUS LOKAL
  -
ALGORITMA
  if isTreeEmpty(p) then { Basis-0 }
    { do nothing }
  else { Rekurens, tidak kosong }
    inOrder(p↑.left)
    proses(p)
    inOrder(p↑.right)
```



Urutan proses = C-B-D-A-F-E-G

Traversal – Post-order

```
procedure postOrder (input p: BinTree)
{ I.S. Pohon p terdefinisi }
{ F.S. Semua node pohon p sudah diproses secara postOrder:
       kiri, kanan, akar }
{ Basis Pohon kosong tidak ada yang diproses }
{ Rekurens Proses subpohon kiri p secara post-order
           Proses subpohon kanan p secara post-order
           Proses akar p; }
KAMUS LOKAL
  -
ALGORITMA
  if isTreeEmpty(p) then { Basis-0 }
    { do nothing }
  else { Rekurens, tidak kosong }
    postOrder(p↑.left)
    postOrder(p↑.right)
    proses(p)
```

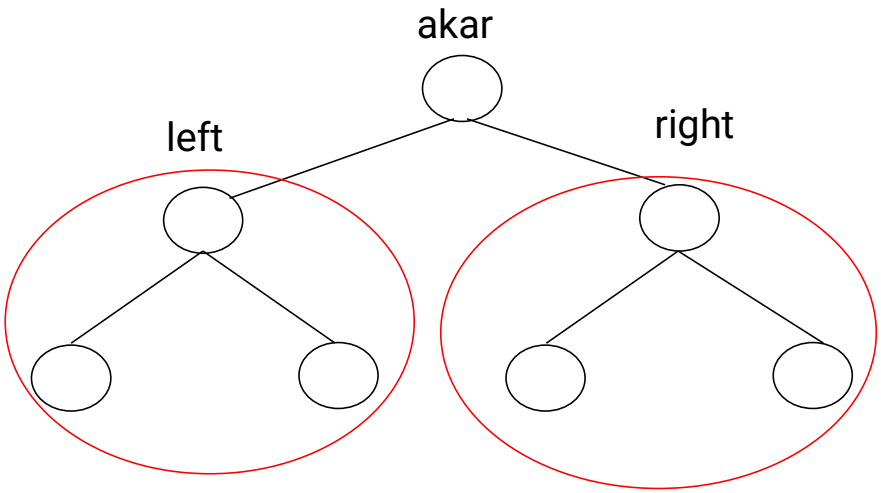


Urutan proses = C-D-B-F-G-E-A

Menghitung Jumlah Elemen

```
function nbElmt (p: BinTree) → integer
{ Mengirim jumlah elemen dari pohon }
```

Berapa jumlah elemen pohon dilihat dari elemen current?



jumlah_elemen =
1 (utk akar) + jumlah_elemen(subpohon kiri) + jumlah_elemen(subpohon kanan)

Menghitung Jumlah Elemen, nbElmt (basis 0)

Rekursif

Basis 0: jika pohon kosong, maka jumlah elemen adalah 0

Rekurens: jumlah elemen = 1 (current element) + jumlah elemen subpohon kiri
+ jumlah elemen subpohon kanan

```
function nbElmt (p: BinTree) → integer
{ Pohon Biner mungkin kosong. Mengirim jumlah elemen dari pohon }
KAMUS LOKAL
  -
ALGORITMA
  if isTreeEmpty(p) then { Basis 0 }
    → 0
  else { Rekurens }
    → 1 + nbElmt(p↑.left) + nbElmt(p↑.right)
```

Menghitung Jumlah Elemen, nbElmt (basis 1)

Rekursif

Basis 1: jika pohon satu elemen, maka jumlah elemen adalah 1

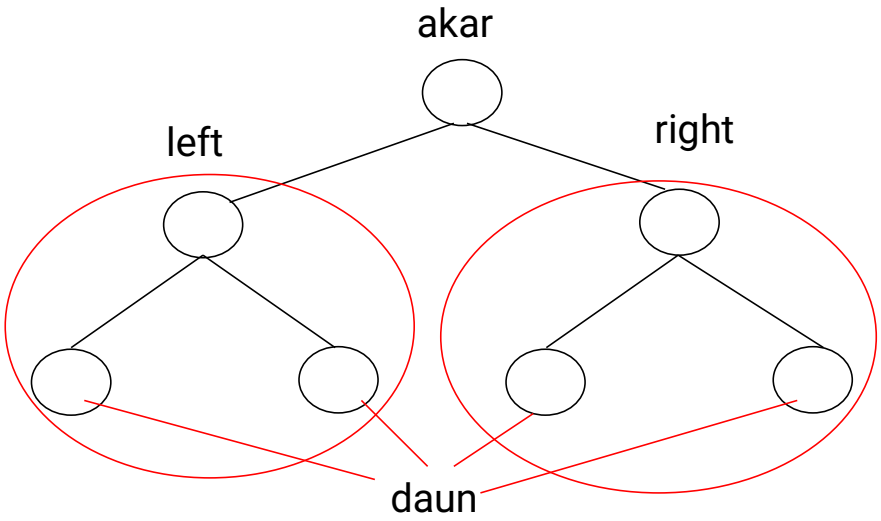
Rekurens: jumlah elemen = 1 (current element) + jumlah elemen subpohon kiri (jika ada)
+ jumlah elemen subpohon kanan (jika ada)

```
function nbElmt (p: BinTree) → integer
{ Pohon Biner tidak kosong. Mengirim jumlah elemen dari pohon }
KAMUS LOKAL
ALGORITMA
  if isOneElmt(p) then { Basis-1 }
    → 1
  else { Rekurens }
    depend on p
      isUnerLeft(p) : → 1 + nbElmt(p↑.left)
      isUnerRight(p): → 1 + nbElmt(p↑.right)
      isBiner(p)    : → 1 + nbElmt(p↑.left) + nbElmt(p↑.right)
```

Menghitung Jumlah Daun

```
function nbLeaf (p: BinTree) → integer
{ Prekondisi: Pohon Biner tidak mungkin kosong.
  Mengirimkan banyaknya daun pohon }
```

Berapa jumlah daun pohon dilihat dari elemen current?



Jumlah daun =
0 (utk akar) + Jumlah_daun(pohon kiri) + Jumlah_daun(pohon kanan)

Menghitung Jumlah Daun, nbLeaf

Analisis Kasus

Pohon kosong jumlah daun = 0

Pohon tidak kosong: jumlah daun dihitung dengan fungsi menghitung jumlah daun dengan **basis-1**

```
function nbLeaf (p: BinTree) → integer
{ Mengirimkan banyaknya daun pohon }
{ Proses perhitungan daun menggunakan nbLeaf basis 1 }
KAMUS LOKAL

ALGORITMA
   if (isTreeEmpty(p)) then
      → 0
   else
      → nbLeaf1(p)
```

nbLeaf1 (Basis-1)

```
function nbLeaf1 (p: BinTree) → integer
{ Prekondisi: Pohon Biner tidak mungkin kosong.
  Mengirimkan banyaknya daun pohon }
{ Basis: Pohon yang hanya mempunyai akar: 1 }
{ Rekurens:
  Punya anak kiri dan tidak punya anak kanan: nbLeaf1(p↑.left)
  Tidak Punya anak kiri dan punya anak kanan: nbLeaf1(p↑.right)
  Punya anak kiri dan punya anak kanan      : nbLeaf1(p↑.left) + nbLeaf1(p↑.right) }
KAMUS LOKAL
  -
ALGORITMA
  if (isOneElmt(p)) then { Basis 1 akar }
    → 1
  else { Rekurens }
    depend on (p)
      isUnerLeft(p) : → nbLeaf1(p↑.left)
      isUnerRight(p): → nbLeaf1(p↑.right)
      isBiner(p)    : → nbLeaf1(p↑.left) + nbLeaf1(p↑.right)
```

Tinggi/Kedalaman Pohon

```
function depth(p: BinTree) → integer
{ Pohon Biner mungkin kosong.
  Mengirim "depth", yaitu tinggi dari pohon }
{ Basis: Pohon kosong, yang mana tingginya nol }
{ Rekurens: 1 + maksimum (depth(anak kiri), depth(anak kanan)) }
KAMUS LOKAL
  -
ALGORITMA
  if isTreeEmpty(p) then { Basis 0 }
    → 0
  else { Rekurens }
    → 1 + max(depth(p↑.left), depth(p↑.right))
```

addLeft

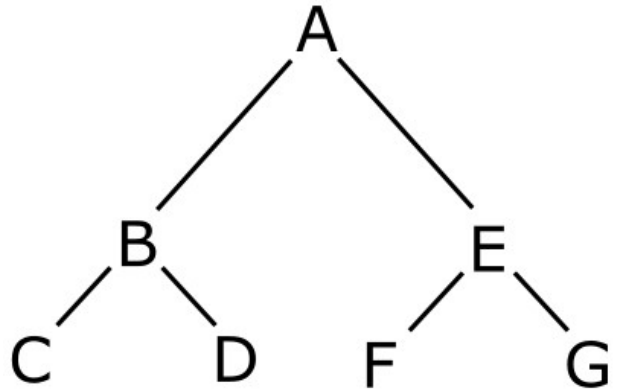
```
procedure addLeft (input/output p: BinTree,
                   input x: ElType)
{ I.S. p boleh kosong }
{ F.S. p bertambah simpulnya, dengan x sebagai simpul daun terkiri }
KAMUS LOKAL
  -
ALGORITMA
  if (isTreeEmpty(p)) then { Basis-0 }
    p ← newTreeNode(x)
  else { Rekurens }
    addLeft(p↑.left,x)
```

DelDaunTerkiri

```
procedure delLeft (input/output p: BinTree,
                           output x: ElType)
{ I.S. p tidak kosong }
{ F.S. Daun terkiri p dihapus, nilai daun ditampung di x }
KAMUS LOKAL
  n: Address
ALGORITMA
  if (isOneElmt(p)) then { Basis-1 }
    x ← p↑.info
    n ← p
    p ← NIL
    deallocTreeNode(n)
  else { Rekurens }
    depend on (p)
      isUnerRight(p): delLeft(p↑.right,x)
      else: delLeft(p↑.left,x)
```

MakeListPreorder

```
function makeListPreorder (p: BinTree) → NodeList
{ Jika p adalah pohon kosong, maka menghasilkan list kosong. }
{ Jika p bukan pohon kosong: menghasilkan list yang elemennya adalah semua elemen
  pohon p dengan urutan Preorder, jika semua alokasi berhasil.
  Menghasilkan list kosong jika ada alokasi yang gagal }
KAMUS LOKAL
  e: AddressList
  l: NodeList
ALGORITMA
  if (isTreeEmpty(p)) then { Basis-0 }
    → NIL
  else { Rekurens }
    e ← newTreeNode(p↑.info)
    if (e ≠ NIL) then
      e↑.next ← makeListPreOrder(p↑.left)
      → concat(e, makeListPreOrder(p↑.right)) {Concat is given}
    else { e gagal dialokasi }
      → NIL
```



Urutan proses = A-B-C-D-E-F-G

Latihan Pohon Biner (1)

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Soal Latihan

```
function search (P: BinTree, X: ElType) → boolean
{ Mengirimkan true jika ada node dari P yang bernilai X }

function isSkewLeft (P: BinTree) → boolean
{ Mengirimkan true jika P adalah pohon condong kiri }

function isSkewRight (P: BinTree) → boolean
{ Mengirimkan true jika P adalah pohon condong kanan }

function level (P: BinTree, X: ElType) → integer
{ Mengirimkan level dari node X yang merupakan salah satu daun dari pohon biner P.
  Akar(P) level-nya adalah 1. Pohon P tidak kosong dan elemen-elemennya unik. }
```

Soal Latihan

```
procedure addDaun (input/Output P: BinTree,
                   input X, Y: ElType,
                   input Kiri: boolean)
{ I.S. P tidak kosong, X adalah daun Pohon Biner P }
{ F.S. P bertambah simpulnya, dengan Y sebagai anak kiri X (jika Kiri), atau sebagai
  anak Kanan X (jika not Kiri). Jika ada lebih dari satu daun bernilai X, Y
  ditambahkan pada daun paling kiri. }

procedure delDaun (input/output P: BinTree,
                   input X: ElType)
{ I.S. P tidak kosong, minimum 1 daun bernilai X }
{ F.S. Semua daun yang bernilai X dihapus dari P }
```

Soal Latihan

```
function makeListDaun (P: BinTree) → ListOfNode
{ Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{ Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua daun
  pohon P. Diasumsikan alokasi selalu berhasil. }

function makeListLevel (P: BinTree, N: integer) → ListOfNode
{ Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{ Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua elemen
  pohon P yang levelnya=N, jika semua alokasi berhasil. Diasumsikan alokasi selalu
  berhasil. }

{CATATAN: gunakan newListNode(ElType) untuk mengalokasi elemen list}
```

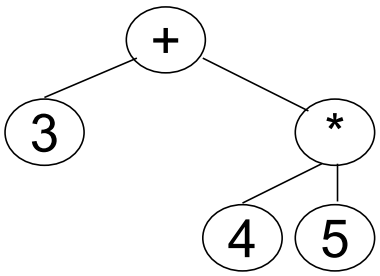
Pohon Biner (Bagian 2)

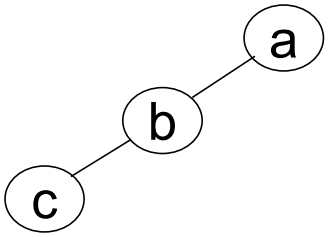
IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

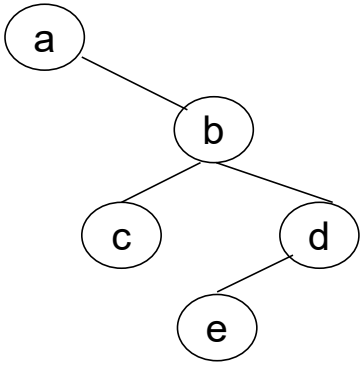
Pohon Biner

Pohon biner adalah himpunan terbatas yang

- mungkin **kosong**, atau
- terdiri atas sebuah simpul yang disebut **akar** dan dua buah himpunan lain yang *disjoint* yang merupakan pohon biner, yang disebut sebagai **sub pohon kiri** dan **sub pohon kanan** dari pohon biner tersebut







Pohon Seimbang

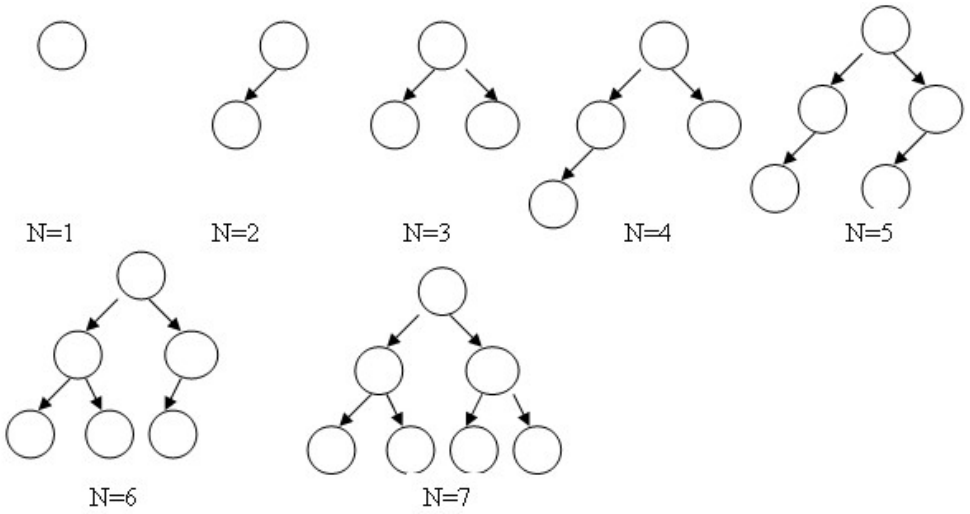
Pohon seimbang (*balanced tree/B-tree*) adalah pohon dengan:

- Perbedaan tinggi subpohon kiri dan subpohon kanan maksimum 1
- Perbedaan banyaknya simpul subpohon kiri dan subpohon kanan maksimum 1
- Subpohon kiri dan subpohon kanan adalah pohon seimbang

Aplikasi: pengelolaan indeks dalam file system dan database system

Yang akan dibahas adalah pohon biner seimbang (balanced binary tree)

Pohon Biner Seimbang



Algoritma untuk membuat pohon biner seimbang dari n buah node

```
function buildBalancedTree (n: integer) → BinTree
{ Menghasilkan sebuah balanced tree }
{ Basis: n = 0: Pohon kosong  }
{ Rekurens: n>0: partisi banyaknya node anak kiri dan kanan,
                 lakukan proses yang sama  }
KAMUS LOKAL
    p: Address; l, r: BinTree; x: ElType
    nL, nR: integer
ALGORITMA
    if (n = 0) then { Basis-0 }
       → NIL
    else { Rekurens }
       { bentuk akar }
       input(x) { mengisi nilai akar }
       p ← newTreeNode(x)
       if (p ≠ NIL) then
          { Partisi sisa node sebagai anak kiri dan anak kanan }
          nL ← n div 2; nR ← n – nL - 1
          l ← buildBalancedTree(nl); r ← buildBalancedTree(nr)
          p↑.left ← l; p↑.right ← r
       → p
```

Binary Search Tree - 1

Binary Search Tree (BST)/pohon biner terurut/pohon biner pencarian adalah pohon biner yang memenuhi sifat:

- Setiap simpul dalam BST mempunyai sebuah nilai
- Subpohon kiri dan subpohon kanan merupakan BST
- Jika p adalah sebuah BST:
  - semua simpul pada subpohon kiri < Akar p
  - semua simpul pada subpohon kanan >= Akar p

Aplikasi BST: algoritma searching dan sorting tingkat lanjut

Binary Search Tree - 2

Nilai simpul (key) dalam BST bisa unik bisa juga tidak.

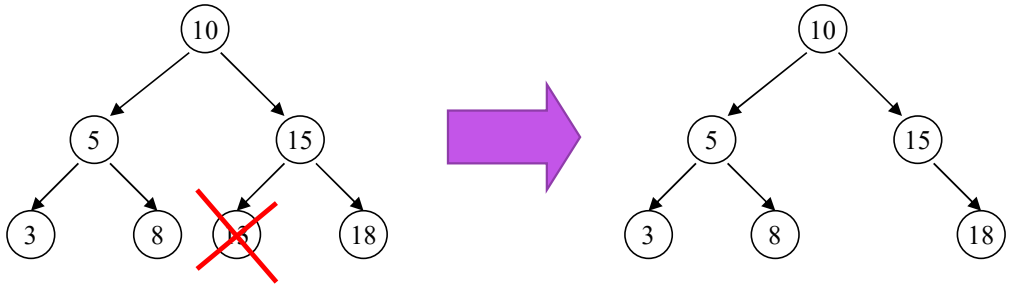
Pada pembahasan ini semua simpul BST (key) bernilai unik. Banyak kemunculan suatu nilai key disimpan dalam field “count”.

```
type ElType: < key: ..., { terdefinisi }
             count: integer >
type Node:  < info: ElType,
              left: BinTree,
             right: BinTree >
type BinTree: Address
```

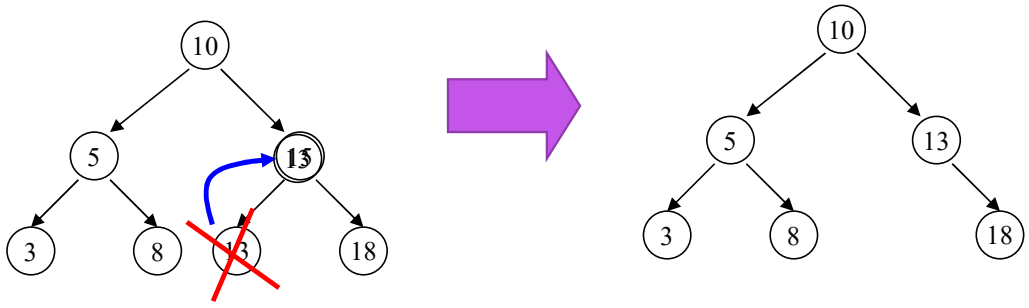
Insert Node dalam BST

```
procedure insSearchTree(input x: ElType, input/output p: BinTree)
{ Menambahkan sebuah node x ke pohon biner pencarian p }
{ ElType terdiri dari key dan count. Key menunjukkan nilai unik, dan count berapa kali muncul }
{ Basis: Pohon kosong  }
{ Rekurens: Jika pohon tidak kosong, insert ke anak kiri jika nilai < p↑.info.key }
{ Atau insert ke anak kanan jika nilai > p↑.info.key }
{ Perhatikan bahwa insert selalu menjadi daun terkiri/terkanan dari subpohon }
KAMUS LOKAL
   -
ALGORITMA
   if (isTreeEmpty(p)) then { Basis: buat hanya akar }
      CreateTree(x,NIL,NIL,p)
   else { Rekurens }
      depend on x, p↑.info.key
         x.key = p↑.info.key : p↑.info.count ← p↑.info.count + 1
         x.key < p↑.info.key : insSearchTree(x, p↑.left)
         x.key > p↑.info.key : insSearchTree(x, p↑.right)
```

Delete Node dalam BST

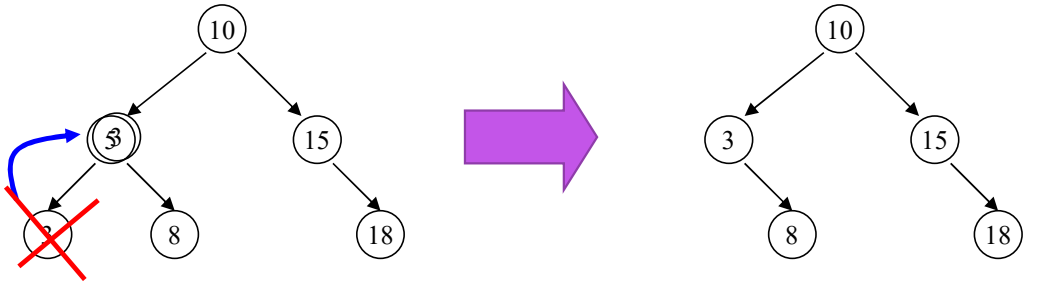


DelBTree(p,13)

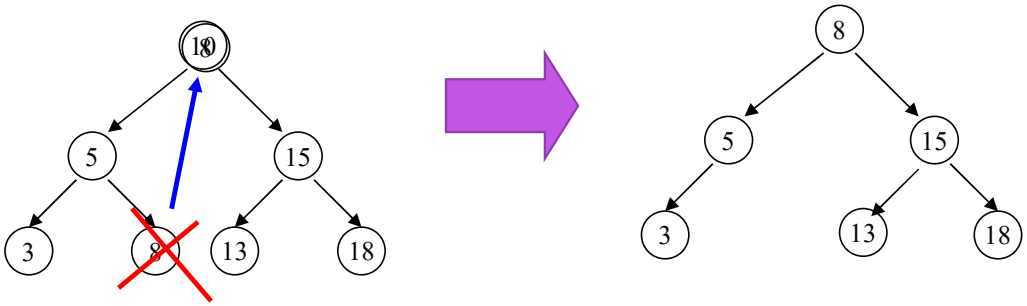


DelBTree(p,15)

Delete Node dalam BST



DelBTree(p,5)



DelBTree(p,10)

Delete Simpul dalam BST - 1

```
procedure delBTree (input/output p: BinTree, input x: ElType)
{ Menghapus simpul bernilai p↑.info.key = x.key, asumsi: x.key pasti ada di p}
{ ElType terdiri dari key dan count. Key menunjukkan nilai unik, dan count berapa kali
  muncul }
{ Basis: ? ; Rekurens: ? }
KAMUS LOKAL
   q: Address
   procedure delNode (input/output p: BinTree)
   { … }
ALGORITMA
   depend on x, p↑.info.key
     x.key < p↑.info.key : delBTree(p↑.left,  x)
     x.key > p↑.info.key : delBTree(p↑.right, x)
     x.key = p↑.info.key : { Delete simpul ini }
                      q ← p
                      depend on q
                         isOneElmt(q)  : p ← NIL
                         isUnerLeft(q) : p ← q↑.left
                         isUnerRight(q): p ← q↑.right
                         isBiner(q)    : delNode(q↑.left)
                      deallocTreeNode(q)
```

Delete Simpul dalam BST - 2

```
procedure delNode (input/output p: BinTree)
{ I.S. p adalah pohon biner tidak kosong }
{ F.S. q berisi salinan nilai daun terkanan }
{ Proses: }
{ Memakai nilai q yang global}
{ Traversal sampai NODE terkanan, copy nilai NODE terkanan p,
  salin nilai ke q semula }
{ q adalah NODE TERKANAN yang akan dihapus }
KAMUS LOKAL
   -
ALGORITMA
   depend on p
      p↑.right ≠ NIL: delNode(p↑.right)
      p↑.right = NIL:       { p anak terkanan }
         q↑.info.key ← p↑.info.key    { salin info p ke q }
         q↑.info.count ← p↑.info.count
         q ← p              { q: yang akan dihapus }
         p ← p↑.left        { pastikan anak kiri "naik", jika ada }
```

Membentuk Pohon Biner dari Pita Karakter

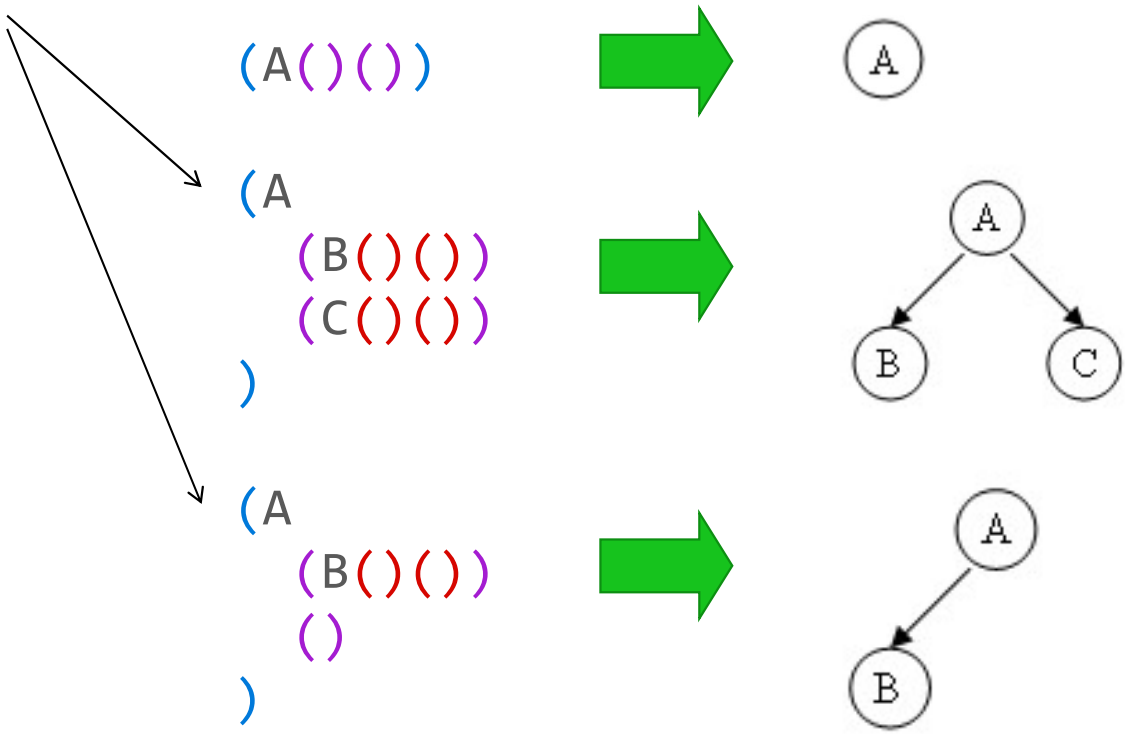
Ekspresi pohon dalam bentuk linier (list) dapat dituliskan dalam sebuah pita karakter

Ada 2 ide:

- Membangun pohon secara iteratif
- Membangun pohon secara rekursif

Contoh - 1

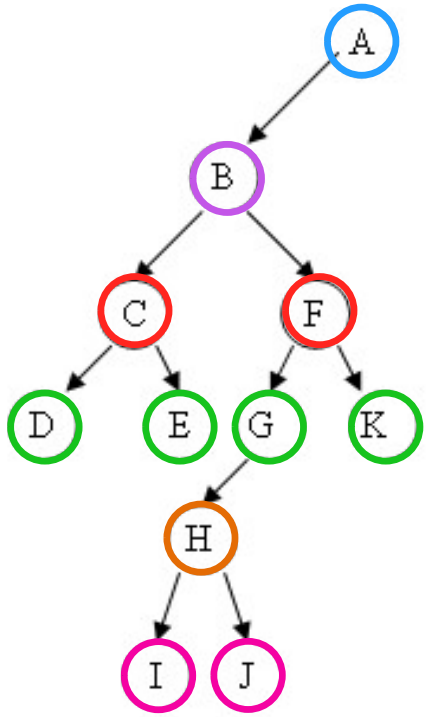
Ditulis dengan ganti baris, berwarna, dan indentasi untuk memudahkan pembacaan



Contoh - 2

Ditulis dengan ganti baris, berwarna, dan indentasi untuk memudahkan pembacaan

```
(A
  (B
    (C
      (D()())
      (E()()))
    (F
      (G
        (H
          (I()())
          (J()()))
        ())
      (K()())))
  ()
)
```



Ide 1: Membangun Pohon secara Iteratif

Karena pembacaan pita dilakukan secara sekuensial, pembentukan pohon selalu dimulai dari akar

Pembacaan karakter demi karakter dilakukan secara iteratif, untuk membentuk sebuah pohon, selalu dilakukan insert terhadap daun

Struktur data memerlukan pointer ke “Bapak”, dengan demikian yang dipakai adalah:

```
type Node: < parent: Address,
             left: Address,
             info: character,
            right: Address >
```

Ide Algoritma Membangun Pohon

Ada tiga kelompok karakter:

- Karakter berupa abjad, menandakan bahwa sebuah node harus dibentuk, entah sebagai anak kiri atau anak kanan.
- Karakter berupa ‘(‘ menandakan suatu sub pohon baru.
  - Jika karakter sebelumnya adalah ‘)’ maka siap untuk melakukan insert sub pohon kanan.
  - Jika karakter sebelumnya adalah abjad, maka siap untuk melakukan insert sub pohon kiri.
- Karakter berupa ‘)’ adalah penutup sebuah pohon, untuk kembali ke “Bapaknya”, berarti naik levelnya dan tidak melakukan apa-apa, tetapi menentukan proses karakter berikutnya.

Tidak cukup dengan mesin karakter (hanya CC), sebab untuk memproses sebuah karakter, dibutuhkan informasi karakter sebelumnya → karena itu digunakan mesin couple (C1, CC)

Implementasi dalam Bahasa C
File: tree.h

```
#include <stdlib.h>
#include "boolean.h"
#include "mesincouple.h"

typedef char ElType;
#define NIL NULL

/*** Selektor ****/
#define   INFO(p) (p)->info
#define   LEFT(p) (p)->left
#define  RIGHT(p) (p)->right
#define PARENT(p) (p)->parent

/*** Type Tree ***/
typedef struct tNode* Address;
typedef struct tNode {
   ElType  info;
   Address left;
   Address right;
   Address parent;
} Node;
typedef Address Tree;
```

Implementasi dalam Bahasa C
File: tree.h

```
Address newTreeNode(ElType x);
/* Alokasi sebuah address p, bernilai tidak NIL jika berhasil */

void CreateTree(Tree *t);
/* I.S. Sembarang */
/* F.S. t terdefinisi */
/* Proses: Membaca isi pita karakter dan membangun pohon dilakukan secara iteratif */

void displayTree (Tree t);
/* I.S. t terdefinisi */
/* F.S. t tertulis di layar */
```

```
void CreateTree (Tree *t) {
  /* Kamus Lokal */
  Address currParent;
  Address ptr;
  int level = 0;
  boolean insKi;
  /* Algoritma */
  startCouple();
  currParent = NIL;
  while (!EOP()) {
    switch (cc) {
      case '(': level++;
                insKi = ( c1 != ')'); //siap sisip kiri
                break;
      case ')': level--;
                if (c1 != '(') {
                  currParent = PARENT(CurrParent);
                }
                break;
      default : ptr = newTreeNode(cc);  /* CC adalah abjad */
                if (currParent != NIL) {
                  if (insKi) { LEFT(currParent) = ptr; }
                  else { RIGHT(currParent) = ptr; }
                } else { *t = ptr; }
                PARENT(ptr) = currParent;
                currParent = ptr;
                break;
    }
    advCouple();
  }
}
```

Implementasi dalam Bahasa C
File: tree.c
Prosedur MakeTree

Ide 2: Membangun Pohon Secara Rekursif - 1

Struktur data yang digunakan adalah tree biasa (tidak memerlukan pointer ke Bapak)

```
typedef struct tNode* Address;
typedef struct tNode {
   ElType  info;
   Address left;
   Address right;
} Node;
typedef Address Tree;
```

Hanya memerlukan modul **mesin karakter** untuk membaca pita karakter

```
void BuildTree(Tree *t)
/* Dipakai jika input dari pita karakter */
/* I.S.: Sembarang */
/* F.S.: T terdefinisi */
/* Proses: Membaca isi pita karakter dan membangun pohon secara
           rekursif */
```

Ide 2: Membangun Pohon secara Rekursif - 2

```
void BuildTree(Tree *t)
/* Dipakai jika input dari pita karakter */
/* I.S. cc berisi '(' */
/* F.S. t terdefinisi */
/* Proses: Membaca isi pita karakter dan membangun pohon secara rekursif, hanya
           membutuhkan mesin karakter */
{   /* Kamus Lokal */

    /* Algoritma */
    adv();                /* advance */
    if (cc==')')          /* Basis: pohon kosong */
        (*t)=NIL;
    else {                /* Rekurens */
        t = newTreeNode(cc);
        adv();            /* advance */
        BuildTree(&(LEFT(*t)));
        BuildTree(&(RIGHT(*t)));
    }
    adv();                /* advance */
}
```

Ide 2: Membangun Pohon secara Rekursif - 3

Contoh pemanggilan di program utama:

```
#include "tree.h"

int main () {
    /* KAMUS */
    Tree t;

    /* ALGORITMA */
    start();
    BuildTree(&t);
    displayTree(t); /* mencetak pohon */
    return 0;
}
```

Membangun Pohon dari String - 1

Menggunakan ide pembangunan pohon dari pita karakter secara rekursif

Struktur data yang digunakan adalah struktur data pohon biasa (tidak perlu pointer ke Bapak)

```
void BuildTreeFromString (Tree *t, char *st, int *idx);
/* Input dari string st */
/* I.S. Sembarang */
/* F.S. t terdefinisi */
/* Proses: Membaca string st dan membangun pohon secara rekursif */
```

Membangun Pohon dari String - 2

```
void BuildTreeFromString (Tree *T, char *st, int *idx)
/* Input dari string st */
/* I.S. st[*idx]=='(' */
/* F.S. T terdefinisi */
/* Proses: Membaca string st dan membangun pohon secara rekursif */
{  /* Kamus Lokal */
   /* Algoritma */
   (*idx)++;                          /* advance */
   if (st[*idx]==')')  /* Basis: pohon kosong */
      (*T)=NIL;
   else {              /* Rekurens */
      t = newTreeNode(st[*idx]);
      (*idx)++;                        /* advance */
      BuildTreeFromString(&LEFT(*t),st,idx);
      BuildTreeFromString(&RIGHT(*t),st,idx);
   }
   (*idx)++;                          /* advance */
}
```

Membangun Pohon dari String - 3

Contoh pemanggilan di program utama:

```
#include "tree.h"

int main () {
    /* KAMUS */
    Tree t;
    char *s = "(A()())";
    int idx = 0;

    /* ALGORITMA */
    BuildTreeFromString(&t,s,&idx);
    displayTree(T); /* mencetak pohon */
    return 0;
}
```

Latihan Soal Pohon Biner (2)

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Soal Latihan

```
function bSearch (p: BinTree, x: ElType) → boolean
{ Mengirimkan true jika ada node dari p (BST) yang bernilai x }

function isBTree (p: BinTree) → boolean
{ Mengirimkan true jika p adalah balanced tree. Asumsi: p tidak kosong }

function buildBST (infos: array [0..99] of integer,
                   nEff: integer) → BinTree
{ Mengirimkan BST yang semua elemennya ada di infos (tidak terurut dan tidak unik).
Infos selalu diisi dari 1. nEff adalah jumlah elemen infos yang terdefinisi, 0 jika
kosong. }
```