

Pohon Biner (Bagian 1)

IF2110 – 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

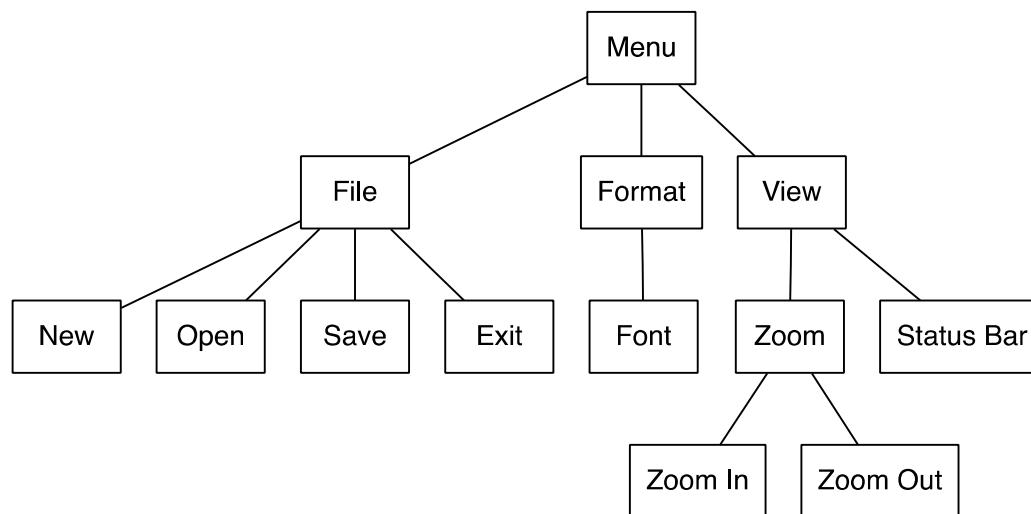
Mahasiswa mampu mengimplementasi fungsi pemroses pohon dalam bahasa C (melalui praktikum)

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

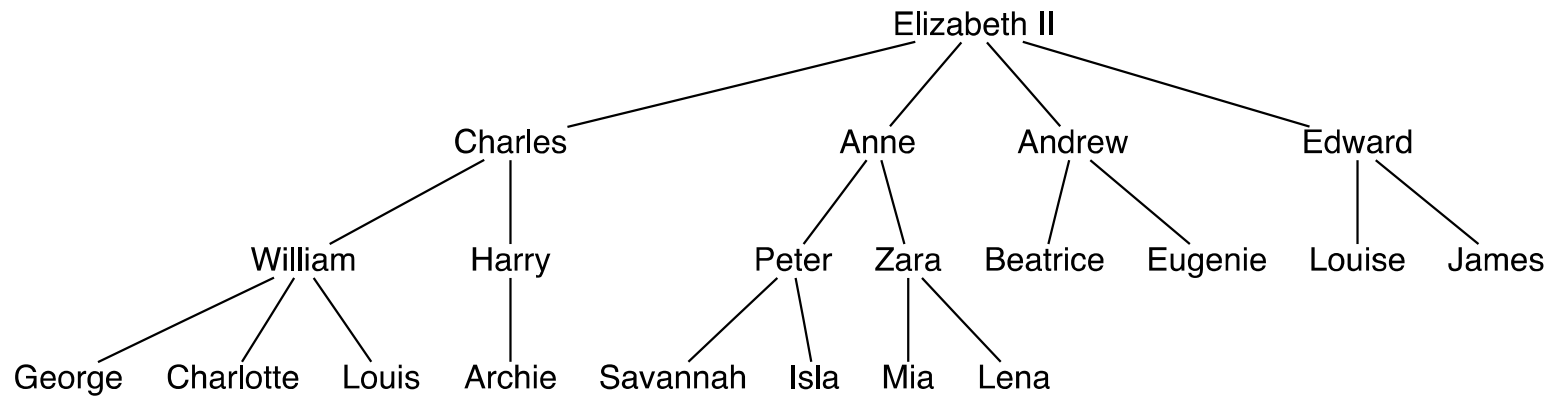
Contoh: Diktat Struktur Data

- 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



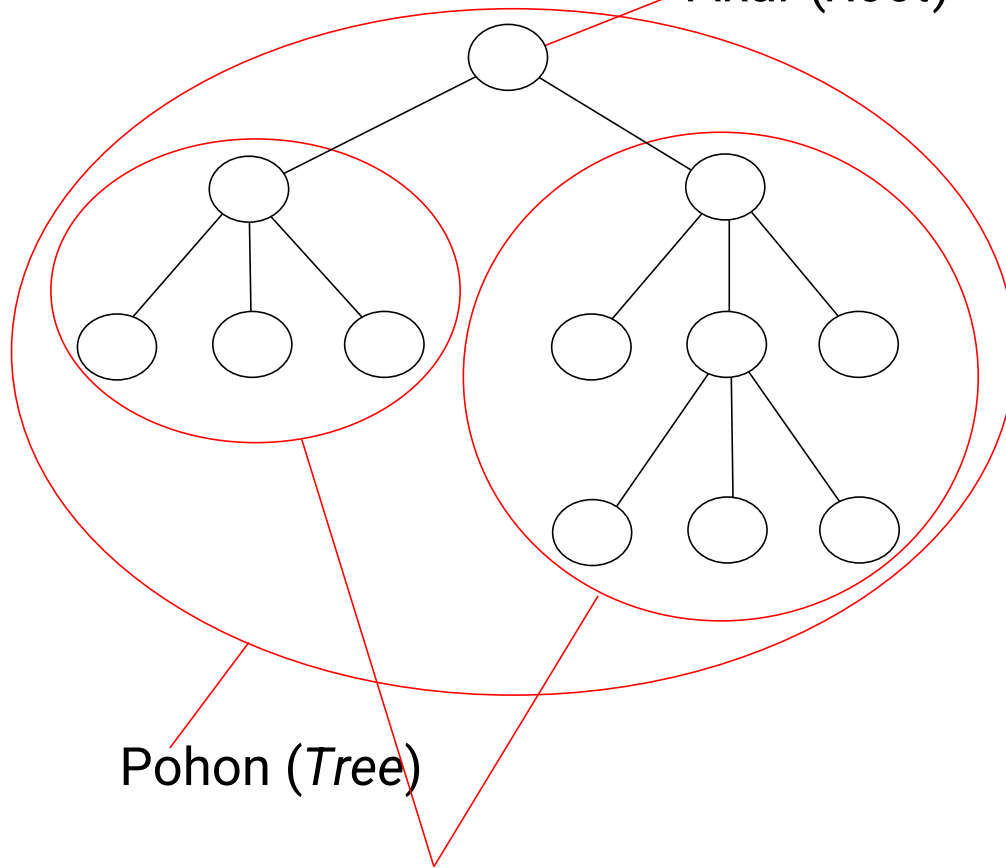
Definisi

Pohon

Akar

Akar (*Root*)

SubPohon



Pohon (*Tree*)

SubPohon (*SubTree*)

Definisi Rekursif Pohon:

- Akar \rightarrow basis
- Sub Pohon (sub himpunan yang berupa pohon)
 \rightarrow rekurens

Definisi Rekursif Pohon

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

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

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

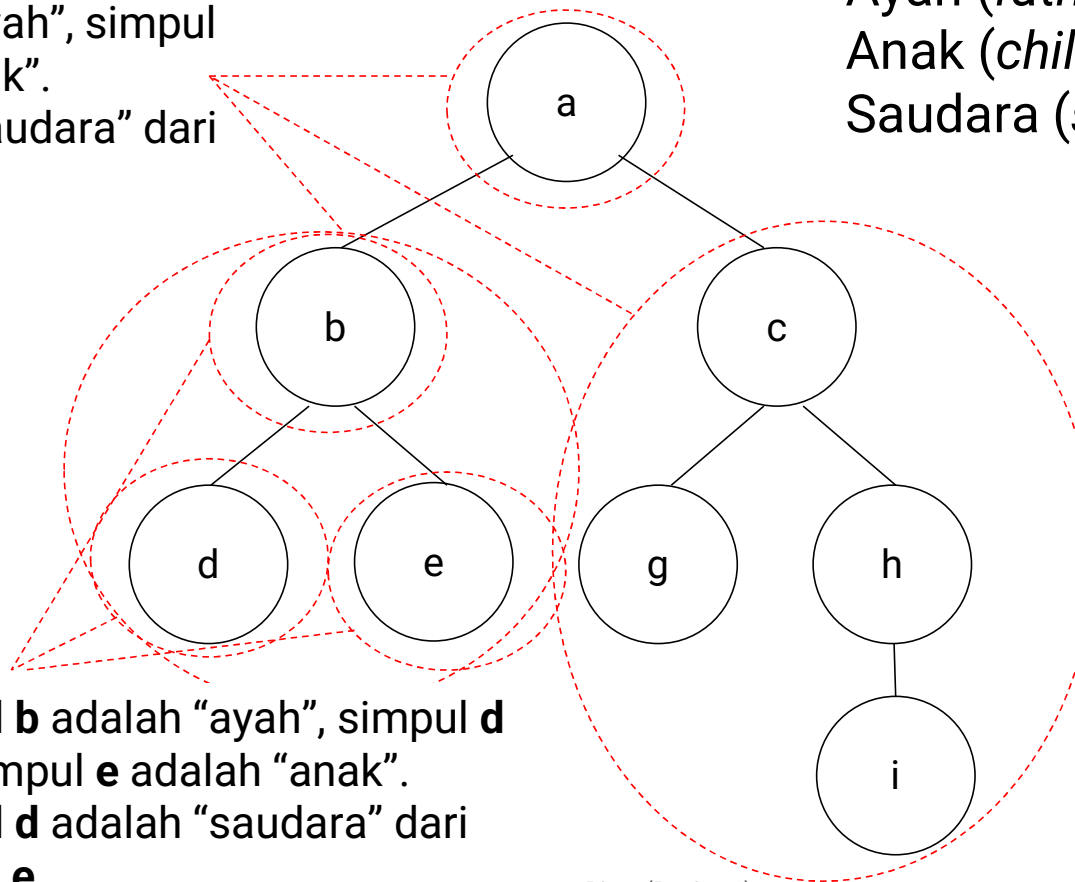
- Binaire (binary), maksimum subpohon: 2
- N-aire, maksimum subpohon: N

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,

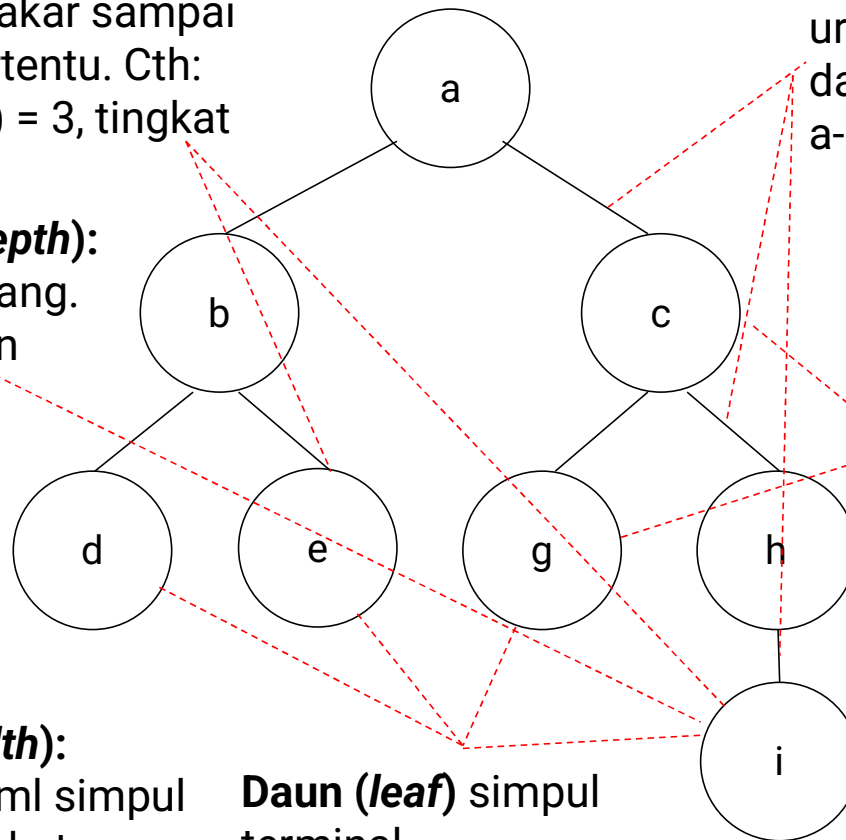
Kedalaman (*depth*): tingkat terpanjang. Cth: kedalaman pohon=4

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

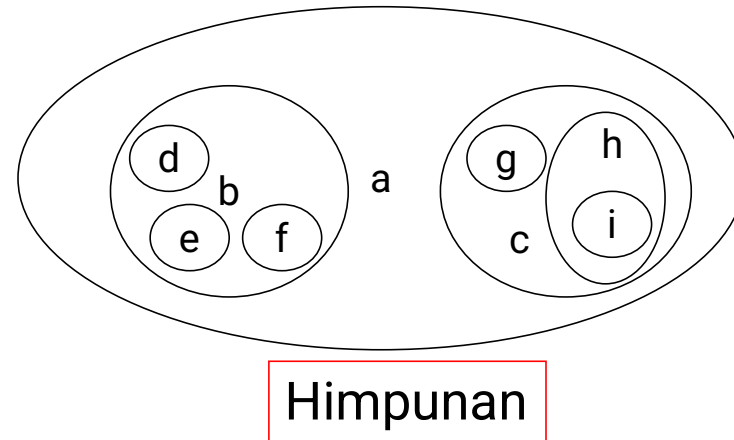
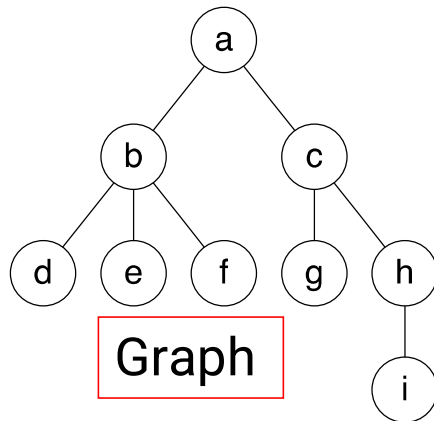
Daun (*leaf*) simpul terminal

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



Beberapa Ilustrasi Representasi



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))))

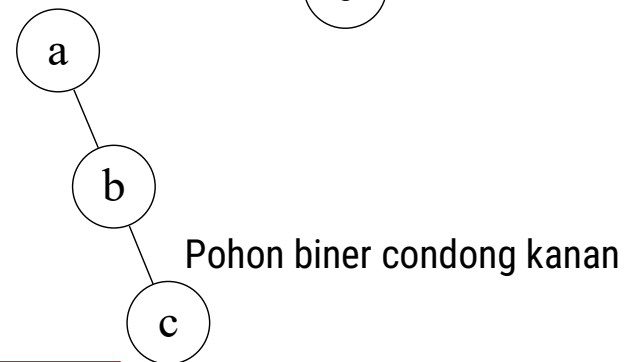
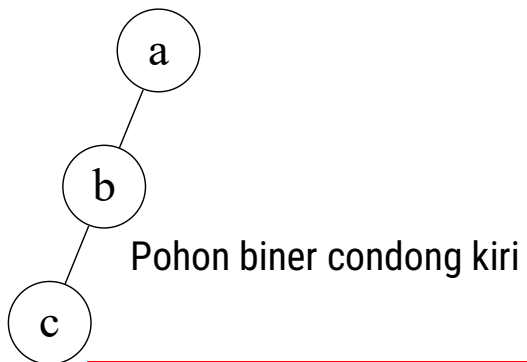
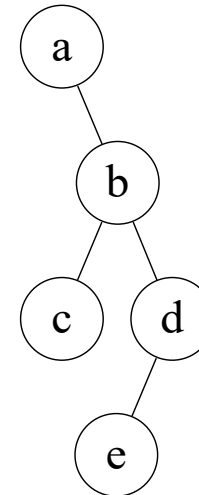
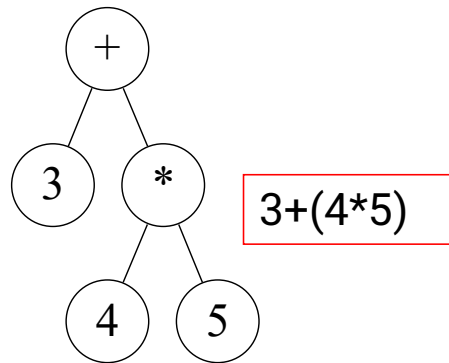
Postfix: (((d, e, f) b, (g, (i) h) c) a)

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



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 isEmpty (p: BinTree) → boolean

{ Mengirimkan true jika p adalah pohon biner yang kosong }

KAMUS LOKAL

-

ALGORITMA

→ (p = NIL)

function hasOneElmt (p: BinTree) → boolean

{ Mengirimkan true jika p tidak kosong dan hanya terdiri atas 1 elemen }

KAMUS LOKAL

-

ALGORITMA

if not(isEmpty(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 **isEmpty**}

Rekursif: 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

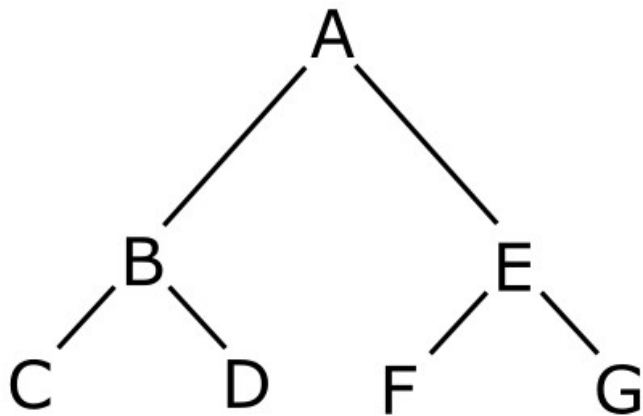
Basis: pohon biner yang hanya terdiri dari akar {menggunakan predikat **isOneElmt**}

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

AU1



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

AU1

Saya tambahkan overview high levelnya dulu sebelum masuk ke masing2 pemrosesan traversal

Ardian Umam; 02/11/2019

Traversal - Preorder

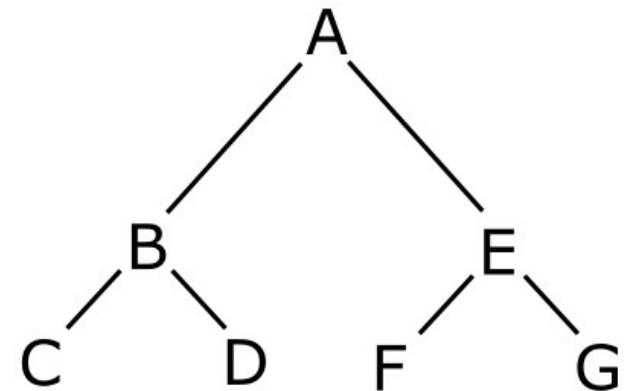
```
procedure preOrder (input p: BinTree)
{ I.S. Pohon p terdefinisi }
{ F.S. Semua node pohon p sudah diproses secara pre-order:
    akar, kiri, kanan }
{ Basis Pohon kosong tidak ada yang diproses }
{ Rekurens Proses akar p;
    Proses subpohon kiri p secara pre-order
    Proses subpohon kanan p secara pre-order }
```

KAMUS LOKAL

-

ALGORITMA

```
if isEmpty(p) then { Basis-0 }
    { do nothing }
else { Rekurens, tidak kosong }
    proses(p)
    preOrder(p↑.left)
    preOrder(p↑.right)
```



Contoh - PrintPreOrder

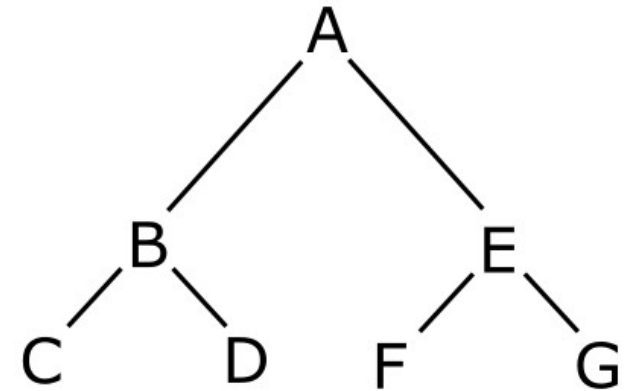
```
procedure printPreOrder (input p: BinTree)
{ I.S. Pohon p terdefinisi }
{ F.S. Semua node pohon p sudah dicetak secara pre-order:
    akar, kiri, kanan }
{ Basis Pohon kosong tidak ada yang diproses }
{ Rekurens Cetak akar p;
    Cetak subpohon kiri p secara pre-order
    Cetak subpohon kanan p secara pre-order }
```

KAMUS LOKAL

-

ALGORITMA

```
if isEmpty(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

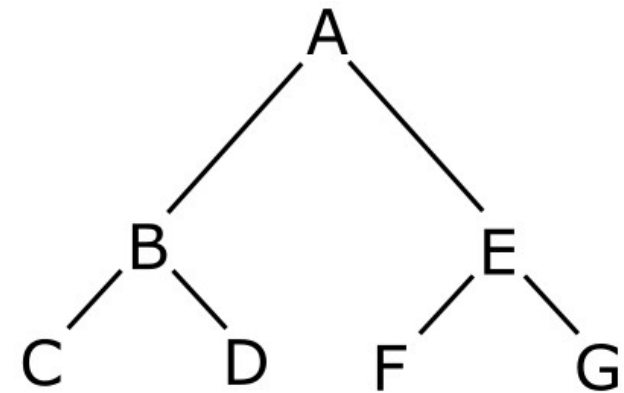
```
procedure inOrder (input p: BinTree)
{ I.S. Pohon p terdefinisi }
{ F.S. Semua node pohon p sudah diproses secara InOrder:
    kiri, akar, kanan }
{ Basis Pohon kosong tidak ada yang diproses }
{ Rekurens Proses subpohon kiri p secara in-order
    Proses akar p;
    Proses subpohon kanan p secara in-order }
```

KAMUS LOKAL

-

ALGORITMA

```
if isEmpty(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

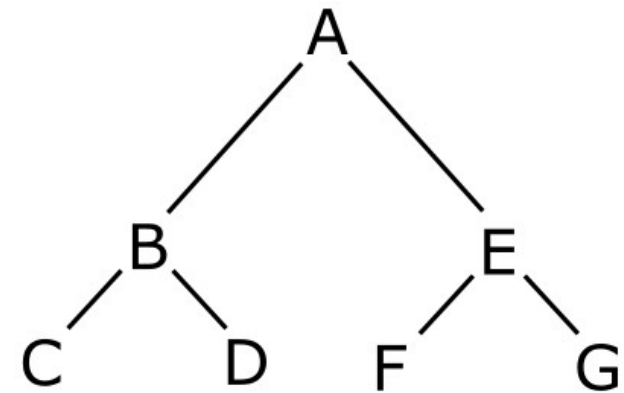
```
procedure postOrder (input p: BinTree)
{ I.S. Pohon p terdefinisi }
{ F.S. Semua node pohon p sudah diproses secara postOrder:
    kiri, kanan, akar }
{ Basis Pohon kosong tidak ada yang diproses }
{ Rekurens Proses subpohon kiri p secara post-order
    Proses subpohon kanan p secara post-order
    Proses akar p; }
```

KAMUS LOKAL

-

ALGORITMA

```
if isEmpty(p) then { Basis-0 }
    { do nothing }
else { Rekurens, tidak kosong }
    postOrder(p↑.left)
    postOrder(p↑.right)
    proses(p)
```

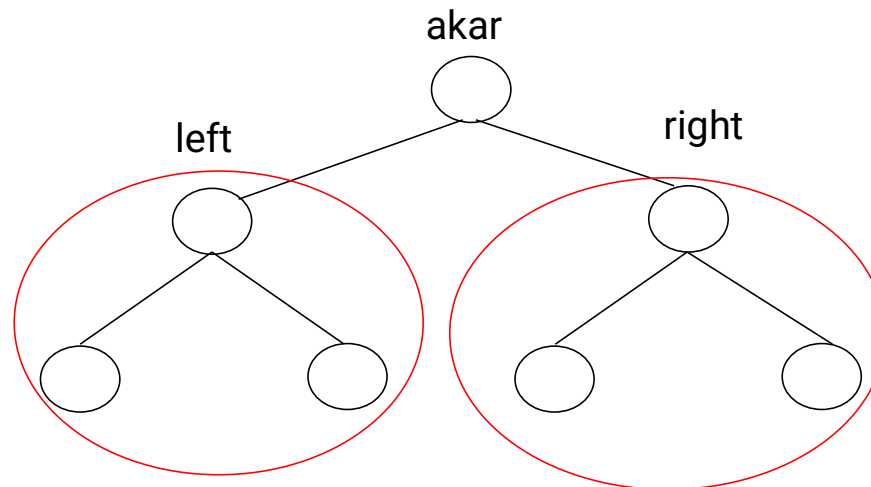


Urutan proses = C-D-B-F-G-E-A

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 isEmpty(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)
```

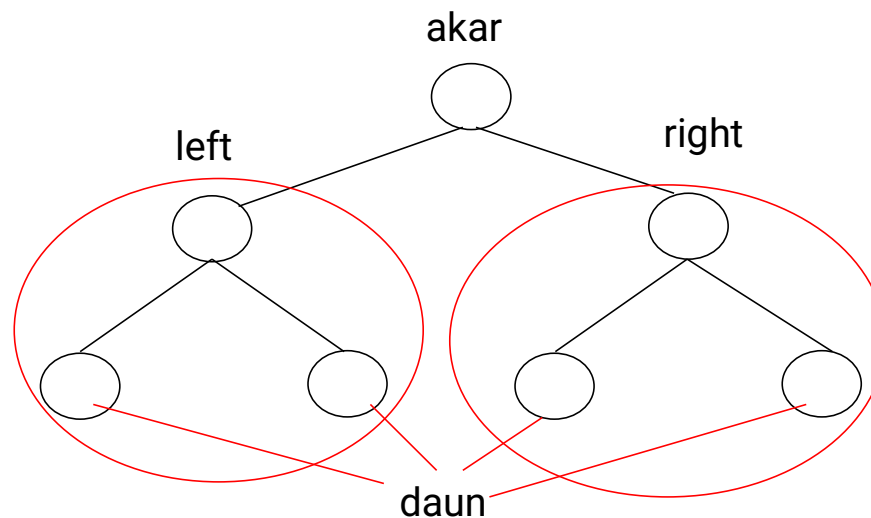
Menghitung Jumlah Daun

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 (isEmpty(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**: $1 + \text{maksimum}(\text{depth}(\text{anak kiri}), \text{depth}(\text{anak kanan}))$ }

KAMUS LOKAL

-

ALGORITMA

if isEmpty(p) then { Basis 0 }

→ 0

else { Rekurens }

→ $1 + \max(\text{depth}(p \uparrow \text{left}), \text{depth}(p \uparrow \text{right}))$

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 (isEmpty(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 (isEmpty(p)) **then** { Basis-0 }

→ NIL

else { Rekurens }

e ← newTreeNode(p↑.info)

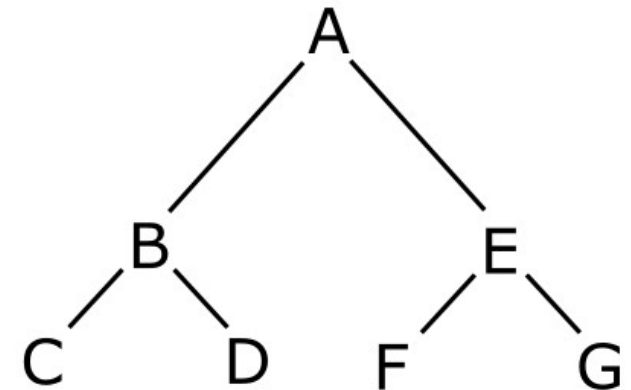
if (e ≠ NIL) **then**

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