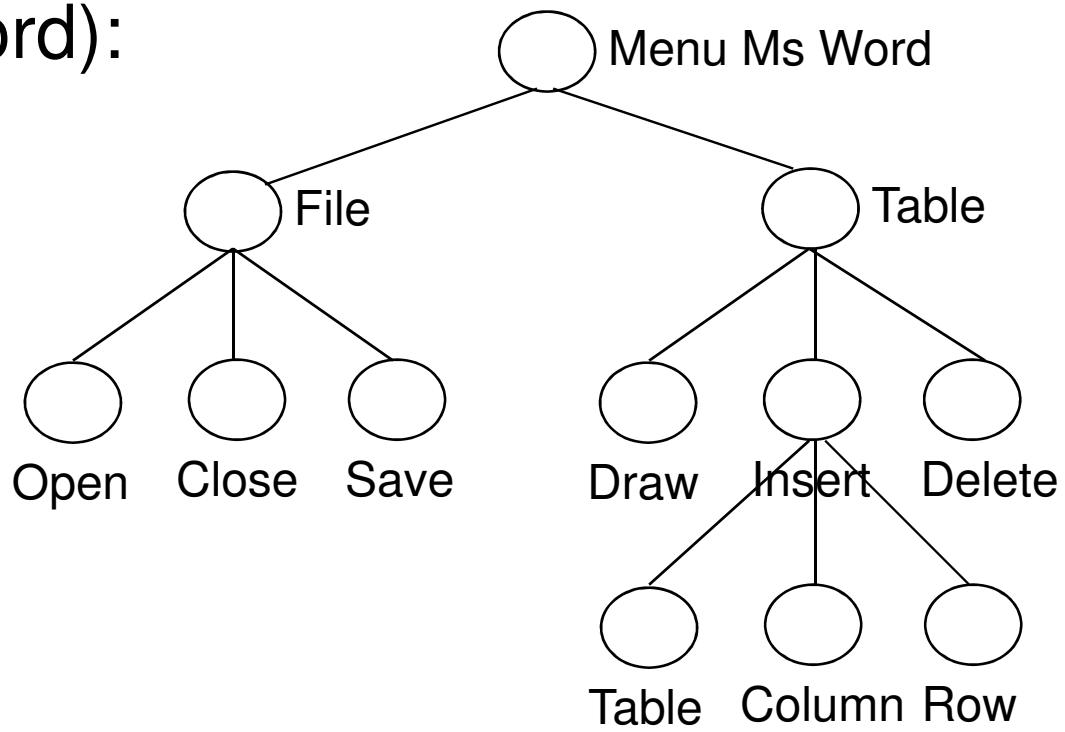
Pohon Biner

Tim Pengajar IF2030

Semester I/2009-2010

Contoh Persoalan - 1

- Menu dalam Aplikasi Komputer
- Contoh (Ms Word):
  - File
    - Open
    - Close
    - Save
  - Table
    - Draw
    - Insert
      - Table
      - Column
      - Row
    - Delete

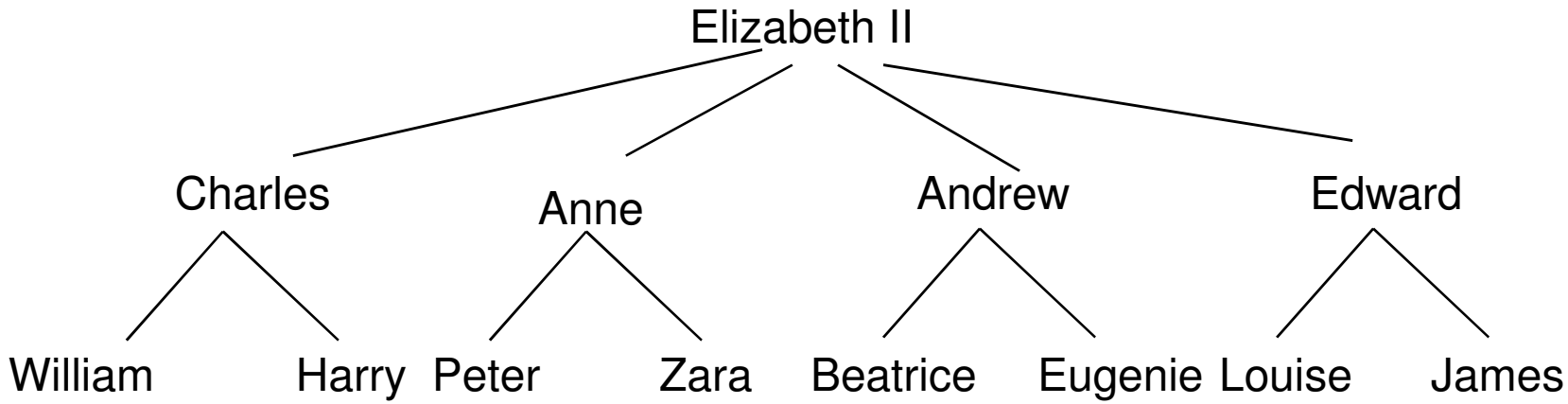


Contoh Persoalan - 2

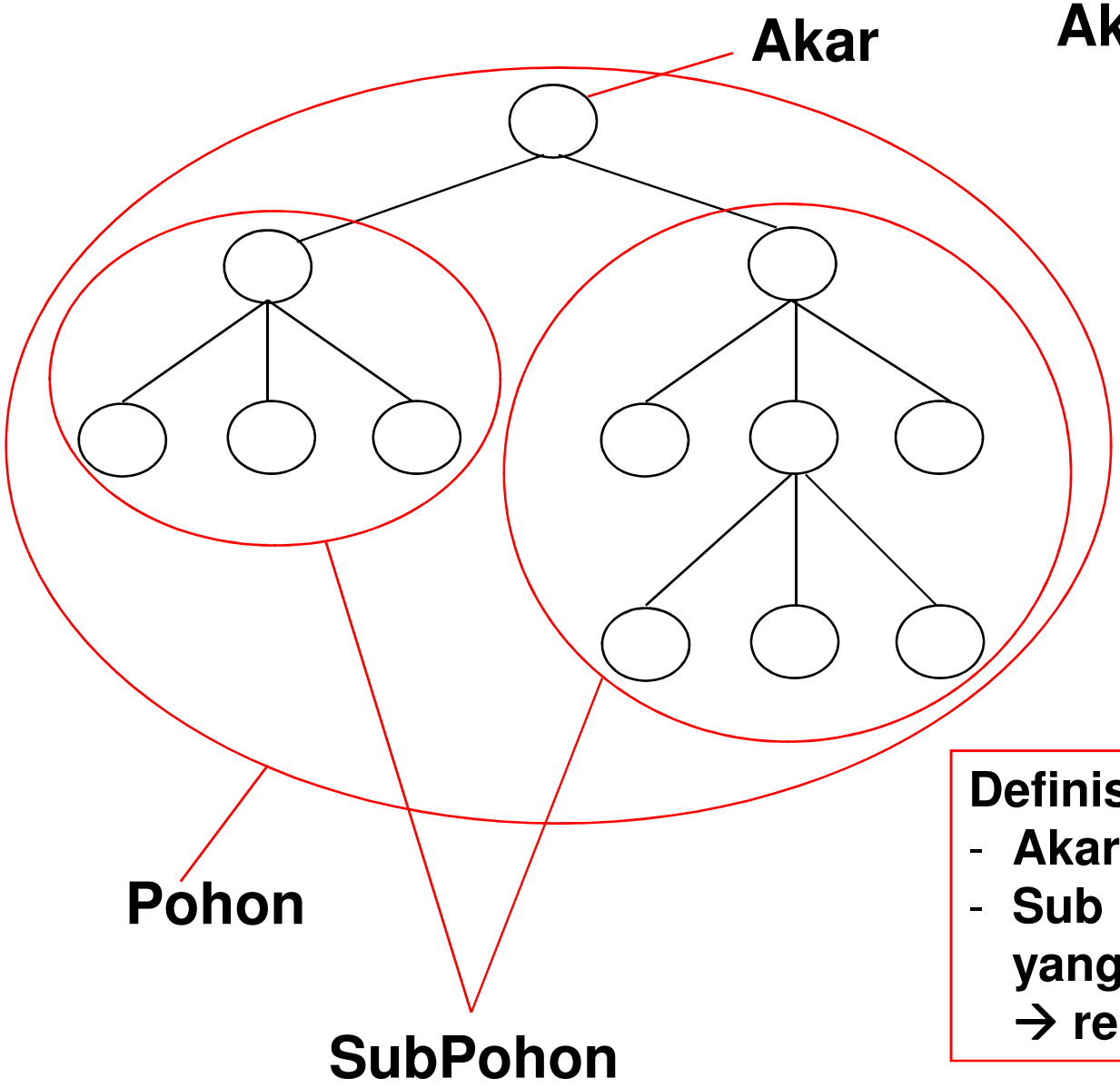
- Susunan bab dalam buku
- Contoh: Diktat Struktur Data
  - Bagian I. Struktur Data
    - Abstract Data Type
      - ADT JAM dalam Bahasa Algoritmik
      - ADT POINT dalam Bahasa Algoritmik
      - ADT GARIS dalam Bahasa Algoritmik
      - Latihan Soal
    - Koleksi Objek
    - …
  - Bagian II
    - 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



Pohon



Akar

SubPohon

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

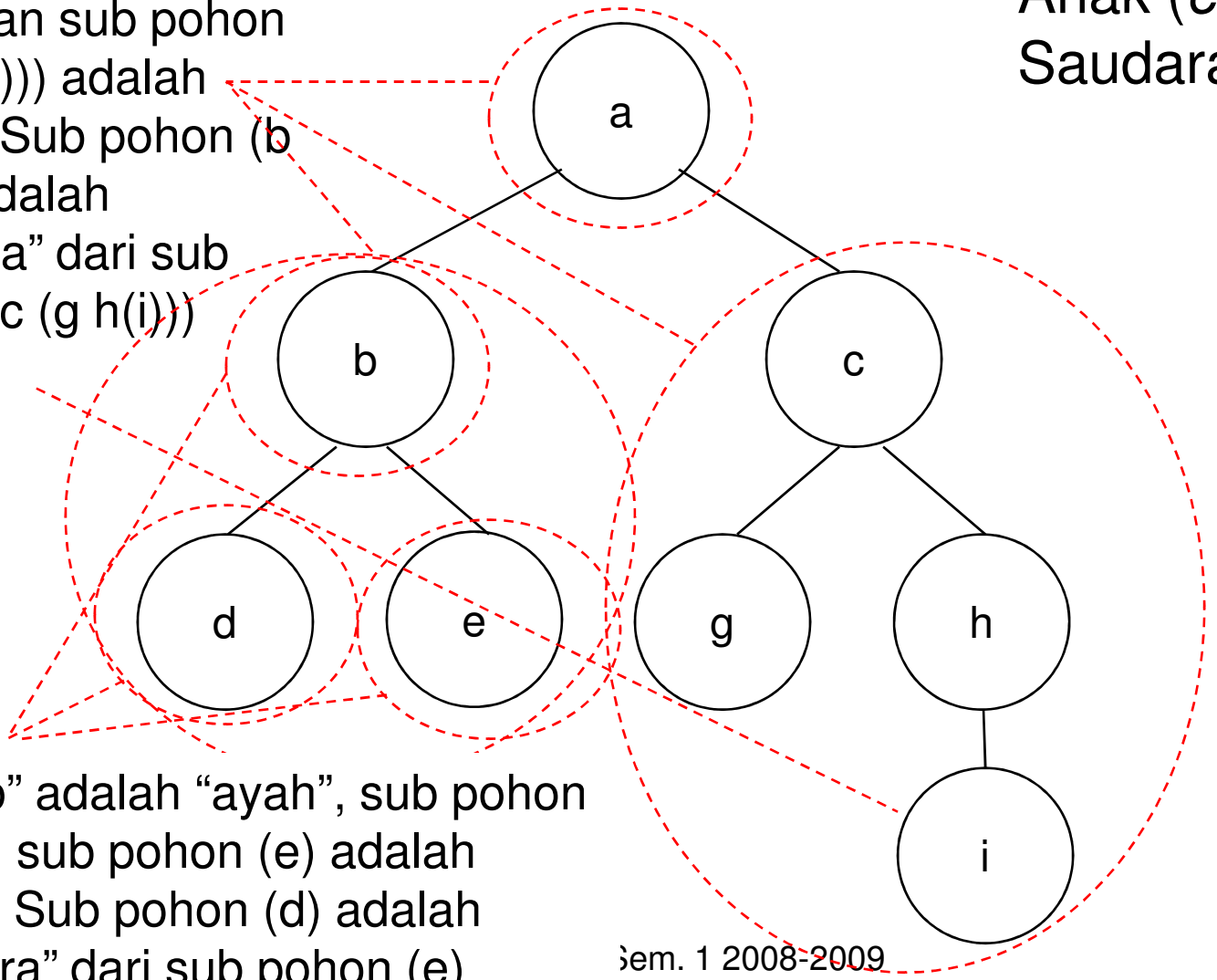
Definisi Rekursif Pohon

- Pohon (tree) adalah himpunan terbatas, tidak kosong, dengan elemen dibedakan 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

Akar “a” adalah “ayah”, sub pohon (b (d e)) dan sub pohon (c (g h(i))) adalah “anak”. Sub pohon (b (d e)) adalah “saudara” dari sub pohon (c (g h(i)))

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



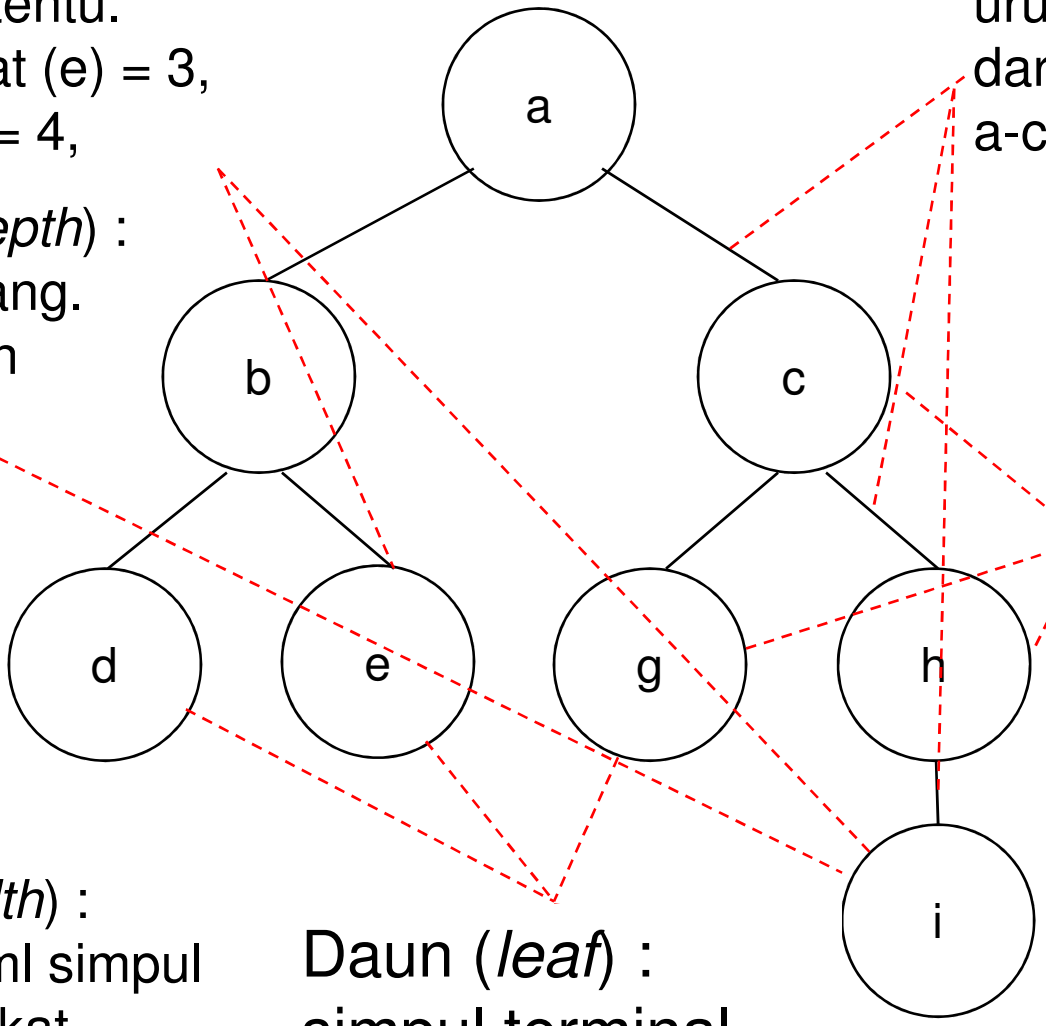
Akar “b” adalah “ayah”, sub pohon (d) dan sub pohon (e) adalah “anak”. Sub pohon (d) adalah “saudara” dari sub pohon (e)

Istilah

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

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

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

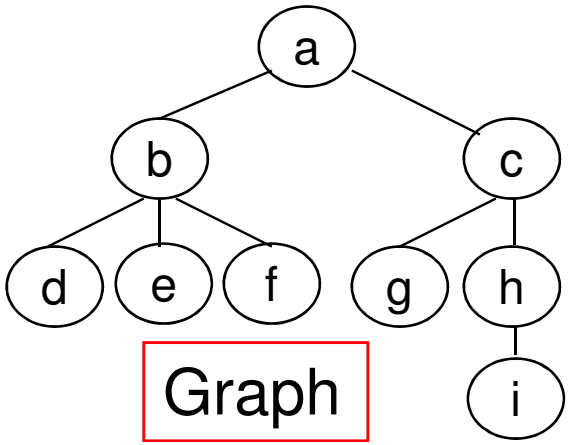


Derajat : banyaknya anak sebuah simpul. Cth, derajat(c)=2, derajat(h)=1, derajat(g)=0

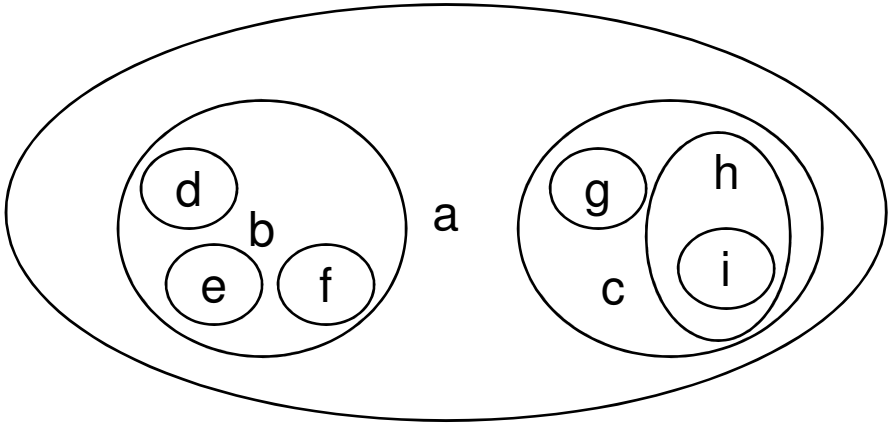
Lebar (*breadth*) : maksimum jml simpul pd suatu tingkat.

Daun (*leaf*) : simpul terminal

Beberapa Ilustrasi Representasi



Graph



Himpunan

Indentasi

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

Bentuk Linier

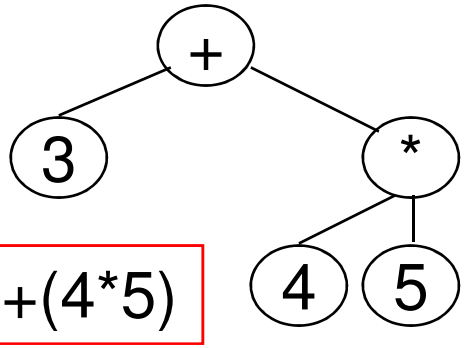
Prefix: (a (b (d (), e (), f ()), c (g (), h (i ()))))
(a (b (d) (e) (f)) (c (g) (h (i))))

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

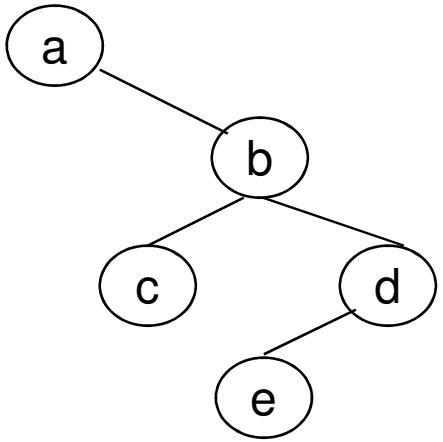
Pohon Biner

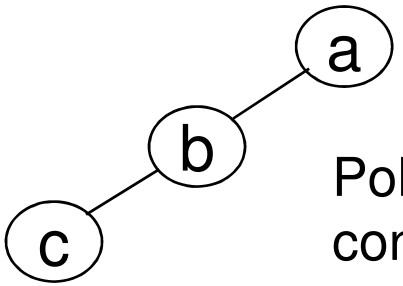
- Pohon biner adalah himpunan terbatas yang
  - mungkin **kosong**, atau
  - terdiri atas sebuah simpul yang disebut **akar** dan dua buah himpunan lain yang *disjoint* yang merupakan pohon biner, yang disebut sebagai **sub pohon kiri** dan **sub pohon kanan** dari pohon biner tersebut
- Perhatikanlah perbedaan pohon biner dengan pohon N-aire:
  - pohon biner mungkin kosong,
  - pohon N-aire tidak mungkin kosong

Contoh Pohon Biner

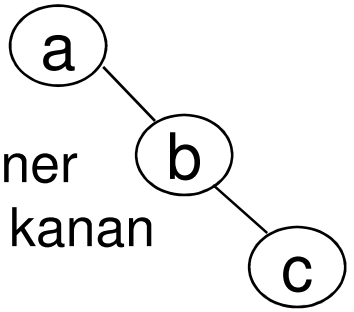


3+(4*5)





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 infotype : ... { terdefinisi }
    type address  : pointer to node
    { Type Pohon Biner }
    type BinTree  : address
    type node     : < Info  : infotype, { simpul/akar }
                      Left  : BinTree,  { subpohon kiri }
                      Right : BinTree   { subpohon kanan } >

{ Tambahan struktur data list untuk pengelolaan elemen pohon }
    type ElmtNode : < Info : infotype,
                      Next : addressList >
    type ListOfNode : addressList
    { list linier yang elemennya adalah ElmtNode }
```

Konstruktor


```
function Tree (Akar : infotype, L : BinTree, R : BinTree)
→ BinTree
{ Menghasilkan sebuah pohon biner dari A, L, dan R, jika
alokasi berhasil }
{ Menghasilkan pohon kosong (Nil) jika alokasi gagal }

procedure MakeTree (input Akar : infotype,
                    input L : BinTree, input R : BinTree,
                    output P : BinTree)
{ I.S. Sembarang }
{ F.S. Menghasilkan sebuah pohon P }
{ Menghasilkan sebuah pohon biner P dari A, L, dan R, jika
alokasi berhasil }
{ Menghasilkan pohon P yang kosong (Nil) jika alokasi
gagal }
```

Selektor

function Akar (P : BinTree) → infotype

{ Mengirimkan nilai Akar pohon biner P }

function Left (P : BinTree) → BinTree

{ Mengirimkan subpohon kiri pohon biner P }

function Right (P : BinTree) → BinTree

{ Mengirimkan subpohon kanan pohon biner P }

Memory Management

function Alokasi (X : infotype) → address
{ Mengirimkan address hasil alokasi sebuah elemen X }
{ Jika alokasi berhasil, maka address tidak nil, dan misalnya menghasilkan P, maka Info(P)=X, Left(P)=Nil, Right(P)=Nil */
{ Jika alokasi gagal, mengirimkan Nil }

procedure Dealokasi (P : address)
{ I.S. P terdefinisi }
{ F.S. P dikembalikan ke sistem }
{ Melakukan dealokasi/pengembalian address P }

Catatan: untuk ListOfNode 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
    → not(IsTreeEmpty(L)) and
      (Left(P) = Nil) and (Right(P) = Nil)
```

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 }

Traversal - Preorder

```
procedure PreOrder (input P : BinTree)
{ I.S. Pohon P terdefinisi }
{ F.S. Semua node pohon P sudah diproses secara PreOrder:
       akar, kiri, kanan }
{ Basis : Pohon kosong : tidak ada yang diproses }
{ Rekurens : Proses Akar(P);
             Proses secara Preorder (Left(P));
             Proses secara Preorder (Right(P)) }
KAMUS LOKAL

ALGORITMA
    if (P = Nil) then { Basis-0 }
       { do nothing }
    else { Rekurens, tidak kosong }
       Proses(P)
       PreOrder(Left(P))
       PreOrder(Right(P))
```

Contoh - PrintPreOrder


```
procedure PrintPreOrder (input P : BinTree)
{ I.S. Pohon P terdefinisi }
{ F.S. Semua node pohon P sudah dicetak secara PreOrder:
       akar, kiri, kanan }
{ Basis : Pohon kosong : tidak ada yang diproses }
{ Rekurens : Cetak Akar(P);
             cetak secara Preorder (Left(P));
             cetak secara Preorder (Right(P)) }
KAMUS LOKAL

ALGORITMA
    if (P = Nil) then { Basis-0 }
       { do nothing }
    else { Rekurens, tidak kosong }
       output(Akar(P))
       PrintPreOrder(Left(P))
       PrintPreOrder(Right(P))
```

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 secara inorder (Left(P));
             Proses Akar(P);
             Proses secara inorder (Right(P)) }
KAMUS LOKAL

ALGORITMA
    if (P = Nil) then { Basis-0 }
       { do nothing }
    else { Rekurens, tidak kosong }
       InOrder(Left(P))
       Proses(P)
       InOrder(Right(P))
```

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 secara PostOrder (Left(P));
             Proses secara PostOrder (Right(P));
             Proses Akar(P); }
KAMUS LOKAL

ALGORITMA
    if (P = Nil) then { Basis-0 }
       { do nothing }
    else { Rekurens, tidak kosong }
       PostOrder(Left(P))
       PostOrder(Right(P))
       Proses(P)
```

Tinggi/Kedalaman Pohon


```
function Tinggi (R : BinTree) → integer
{ Pohon Biner mungkin kosong. Mengirim "depth" yaitu
tinggi dari pohon }
{ Basis: Pohon kosong: tingginya nol }
{ Rekurens: 1 + maksimum (Tinggi(Anak kiri),
Tinggi(AnakKanan)) }
KAMUS LOKAL

ALGORITMA
    if (R = Nil) then { Basis 0 }
       → 0
    else { Rekurens }
       → 1 + max (Tinggi(Left(R)), Tinggi(Right(R)))
```

NBDaun (Basis-1)


```
function NBDaun (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: NBDaun(Left(P))
  Tidak Punya anak kiri dan punya anak kanan : NBDaun(Right(P))
  Punya anak kiri dan punya anak kanan : NBDaun(Left(P)) +
                                         NBDaun(Right(P))       }
KAMUS LOKAL

ALGORITMA
   if (IsOneElmt(P)) then { Basis 1 : akar }
      → 1
   else { Rekurens }
      depend on (P)
        IsUnerLeft(P)  : → NBDaun(Left(P))
        IsUnerRight(P) : → NBDaun(Right(P))
        IsBiner(P)     : → NBDaun(Left(P))+NBDaun(Right(P))
```

Latihan-Latihan

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

function NbElmt (P : BinTree) → integer
{ Mengirimkan banyaknya elemen (node) pohon biner P }

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 : infotype) → 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. }

Latihan-Latihan


```
procedure AddDaunTerkiri (input/output P : BinTree,
                          input X : infotype)
{ I.S. P boleh kosong }
{ F.S. P bertambah simpulnya, dengan X sebagai simpul daun
terkiri }

procedure AddDaun (input/Output P : BinTree,
                   input X, Y : infotype, input Kiri : boolean)
{ I.S. P tidak kosong, X adalah salah satu 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) }

procedure DelDaunTerkiri (input/output P : BinTree,
                          output X : infotype)
{ I.S. P tidak kosong }
{ F.S. P dihapus daun terkirinya, dan didealokasi, dengan X
adalah info yang semula disimpan pada daun terkiri yang dihapus }

procedure DelDaun (input/output P : BinTree, input X : infotype)
{ I.S. P tidak kosong, X adalah salah satu daun }
{ F.S. X dihapus dari P }
```

Latihan-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, jika semua alokasi berhasil. Menghasilkan list kosong jika ada alokasi yang gagal }

function MakeListPreorder (P : BinTree) → ListOfNode

{ 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 }

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. Menghasilkan list kosong jika ada alokasi yang gagal. }

Search


```
function Search (P : BinTree, X : infotype) → boolean
{ Mengirimkan true jika ada node dari P yang bernilai X }
KAMUS LOKAL

ALGORITMA
   if (IsTreeEmpty(P)) then { Basis-0 }
      → false
   else { Rekurens }
      if (Akar(P) = X) then
         → true
      else
         → Search(Left(P),X) or Search(Right(P),X)
```

NbElmt


```
function NbELmt (P : BinTree) → integer
{ Mengirimkan banyaknya elemen (node) pohon biner P
}
KAMUS LOKAL

ALGORITMA
   if (IsTreeEmpty(P)) then { Basis-0 }
      → 0
   else { Rekurens }
      → 1 + NbElmt(Left(P)) + NbElmt(Right(P))
```

IsSkewLeft - 1


```
function IsSkewLeft (P : BinTree) → boolean
{ Mengirimkan true jika P adalah pohon condong kiri }
{ Asumsi : Pohon kosong adalah pohon condong kiri }
KAMUS LOKAL

ALGORITMA
   if (IsTreeEmpty(P)) then { Basis-0 }
      → true
   else { Rekurens }
      if (not(IsTreeEmpty(Right(P)))) then
          → false
      else
          → IsSkewLeft(Left(P))
```

IsSkewLeft - 2


```
function IsSkewLeft (P : BinTree) → boolean
{ Mengirimkan true jika P adalah pohon condong kiri }
{ Asumsi : Pohon kosong bukan pohon condong kiri, pohon 1
elemen adalah pohon condong kiri }
KAMUS LOKAL

ALGORITMA
   if (IsTreeEmpty(P)) then { Kasus khusus }
      → false
   else
      if (IsOneElmt(P)) then { Basis-1 }
          → true
      else { Rekurens }
          if (IsUnerLeft(P)) then
              → IsSkewLeft(Left(P))
          else { P punya pohon kanan }
              → false
```

Level


```
function Level (P : BinTree, X : infotype) → 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. }
KAMUS LOKAL

ALGORITMA
   if (IsOneElmt(P)) then { Basis-1 }
      → 1 { Akar(P) dijamin pasti X }
   else { Rekurens }
      depend on (P)
         IsUnerLeft(P)  : → 1 + Level(Left(P),X)
         IsUnerRight(P) : → 1 + Level(Right(P),X)
         IsBiner(P)     : if (Search(Left(P),X)) then
                             → 1 + Level(Left(P),X)
                          else { pasti ada di pohon kanan }
                             → 1 + Level(Right(P),X)
```

AddDaunTerkiri

```
procedure AddDaunTerkiri (input/output P : BinTree,
                          input X : infotype)
{ I.S. P boleh kosong }
{ F.S. P bertambah simpulnya, dengan X sebagai
simpul daun terkiri }
KAMUS LOKAL
   N : address
ALGORITMA
   if (IsTreeEmpty(P)) then { Basis-0 }
      N ← Alokasi(X)
      if (N ≠ Nil) then {jika alokasi sukses}
            P ← N
      { else: P tetap }
   else { Rekurens }
      AddDaunTerkiri(Left(P), X)
```

AddDaun


```
procedure AddDaun (input/output P : BinTree,
                   input X, Y : infotype,
                   input Kiri : boolean)
{ I.S. P tidak kosong, X adalah salah satu daun Pohon
Biner P. Tambahan asumsi : X bisa lebih dari satu daun }
{ F.S. P bertambah simpulnya, dengan Y sebagai anak kiri X
(jika Kiri), atau sebagai anak Kanan X (jika not Kiri) }
KAMUS LOKAL
   N : address
ALGORITMA
   if (IsOneElmt(P)) then { Basis-1 }
      if (Akar(P) = X) then
         N ← Alokasi(Y)
         if N ≠ Nil then
            if Kiri then Left(P) ← N
            else { not Kiri } Right(P) ← N
   else { Rekurens }
      depend on (P)
          IsUnerLeft(P)  : AddDaun(Left(P),X,Y,Kiri)
          IsUnerRight(P) : AddDaun(Right(P),X,Y,Kiri)
          IsBiner(P)     : AddDaun(Left(P),X,Y,Kiri)
                           AddDaun(Right(P),X,Y,Kiri)
```

MakeListPreorder

```
function MakeListPreorder (P : BinTree) → ListOfNode
{ 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 : ListOfNode
ALGORITMA
   if (IsTreeEmpty(P)) then { Basis-0 }
      → Nil
   else { Rekurens }
      E ← AlokList(Akar(P))
      if (E ≠ Nil) then
         Next(E) ← MakeListPreOrder(Left(P))
         L ← MakeListPreOrder(Right(P))
         → Concat1 (E,L)
      else { E gagal dialokasi }
         → Nil
```

MakeListDaun


```
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, jika semua alokasi berhasil. Menghasilkan list kosong jika ada
alokasi yang gagal }
KAMUS LOKAL
   E : addressList
   L : ListOfNode
ALGORITMA
   if (IsOneEmpty(P)) then { kasus khusus }
      → Nil
   else { Rekurens }
      if IsOneElmt(P) then  { basis-1 }
          E ← AlokList(Akar(P))
          if (E ≠ Nil) then
             → E
          else { E gagal dialokasi }
             → Nil
      else { Rekurens }
          depend on (P)
               IsUnerLeft(P)  : → MakeListDaun(Left(P))
               IsUnerRight(P) : → MakeListDaun(Right(P))
               IsBiner(P)     : → Concat1(MakeListDaun(Left(P)),
                                          MakeListDaun(Right(P)))
```

Concat1

```
function Concat1 (L1, L2 : ListOfNode) → ListOfNode
{ Menghasilkan gabungan L1 dan L2, tanpa alokasi elemen
   baru }
KAMUS LOKAL
   Last : addressList
ALGORITMA
   if (not(ListEmpty(L1))) then
      Last ← L1
      while (Next(Last) ≠ Nil) then { Basis-0 }
          Last ← Next(Last)
      { Next(Last) = Nil }
      Next(Last) ← L2
      → L1
  else
      → L2
```

PR

- Modul P-15. ADT Pohon Biner (untuk bagian Pohon Seimbang dan Binary Search Tree dikerjakan setelah materi tersebut diberikan)