



Tree (Pohon)

Praktikum Dasar Pemrograman
Pertemuan Ke-11

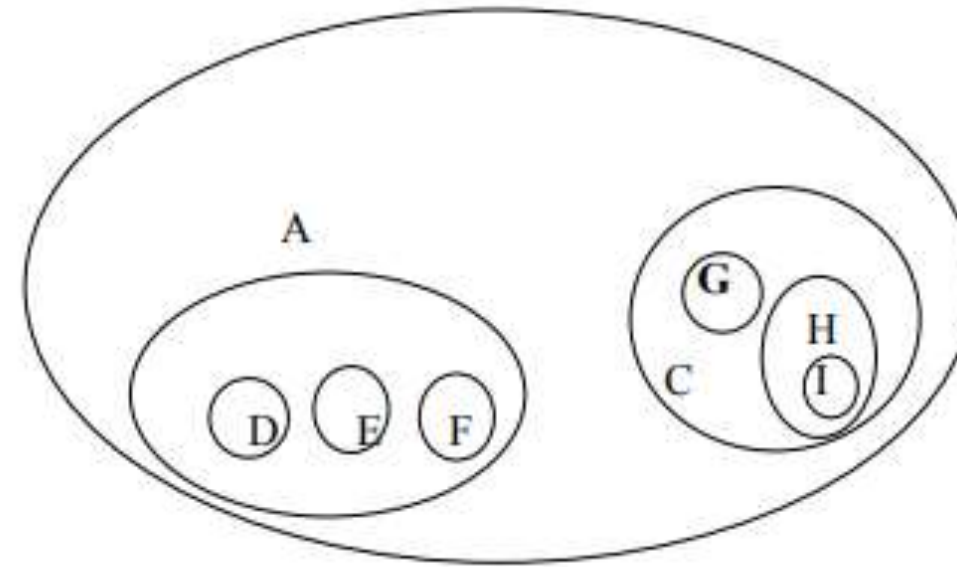
Definisi Rekursif Pohon

Definisi Rekursif: sebuah POHON adalah himpunan terbatas tidak kosong, dengan elemen yang dibedakan sebagai berikut :

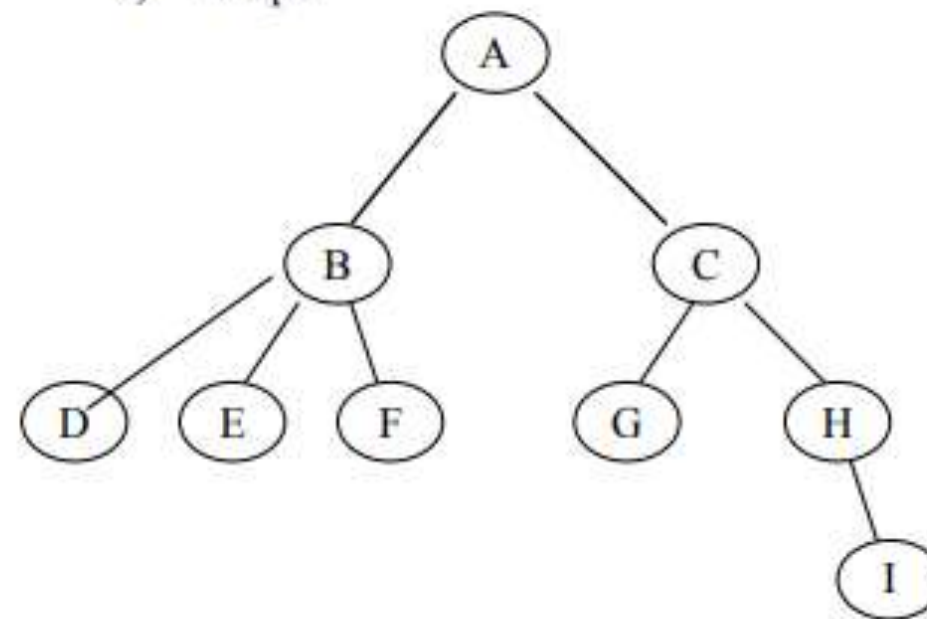
- sebuah elemen dibedakan dari yang lain, yang disebut sebagai AKAR dari pohon.
- elemen yang lain (jika masih ada) dibagi-bagi menjadi beberapa sub himpunan yang disjoint, dan masing-masing sub himpunan tersebut adalah POHON yang disebut sebagai SUB POHON dari pohon yang dimaksud

Cara Penulisan Pohon

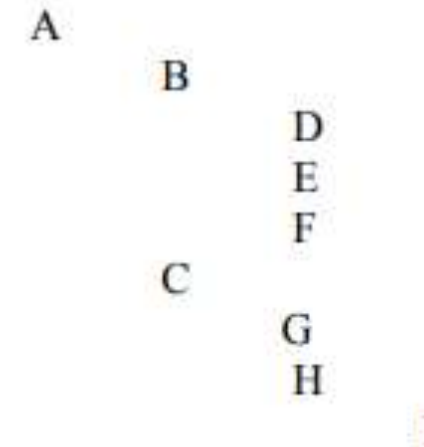
a) Himpunan yang saling melingkupi



b) Graph



c) Indentasi



d) Bentuk linier :

Prefix : (A (B(D(),E(),F()), C(G(),H(I())))), atau
(A(B(D)(E)(F))(C(G)(H(I))))
Postfix : ((D,E,F)B,(G,(I) H) C)

BEBERAPA ISTILAH DALAM TREE

- Hutan (forest)
- Simpul (node, elemen)
- Cabang (path)
- Ayah (father)
- Anak (child)
- Saudara (sibling)
- Daun (leaf)
- Jalan (path)
- Derajat
- Tingkat (level)
- Kedalaman (depth)
- Lebar (breadth)



Pohon N-aire

Pohon N-aire adalah pohon yang pada setiap level anaknya boleh berbeda-beda jumlahnya, dan anaknya tersebut adaalah pohon N-aire

Definisi rekursif

- Basis-1 : pohon yang hanya terdiri dari akar adalah pohon N-aire
- Rekurens : Sebuah pohon N-aire terdiri dari akar dan sisanya ("anak-anak"nya) adalah list pohon N-aire.

TYPE POHON-N-AIRE (tidak mungkin kosong)

DEFINISI DAN SPESIFIKASI TYPE

type Elemen : { tergantung type node }

type PohonN-ner : $\langle A : \text{Elemen}, PN : \text{PohonN-ner} \rangle \{ \text{notasiPrefix} \}$, atau

type PohonN-ner : $\langle PN : \text{PohonN-ner}, A : \text{Elemen} \rangle \{ \text{notasi postfix} \}$

{Pohon N-ner terdiri dari Akar yang berupa elemen dan list dari pohon N-aire yang menjadi anaknya List anak mungkin kosong, tapi pohon N-ner tidak pernah kosong, karena minimal mempunyai sebuah elemen sebagai akar pohon}

DEFINISI DAN SPESIFIKASI SELEKTOR

Akar : PohonN-ner tidak kosong \rightarrow Elemen

{ Akar(P) adalah Akar dari P. Jika P adalah (A,PN) = Akar(P) adalah A }

Anak : PohonN-ner tidak kosong \rightarrow list of PohonN-ner

{ Anak(P) adalah list of pohon N-ner yang merupakan anak-anak (sub phon) dari P. Jika P adalah (A, PN) = Anak (P) adalah PN }

```
1  def makeTreeN(A,PN):
2      return [A,PN]
3
4  def Akar(P):
5      return P[0]
6
7  def Anak(P):
8      return P[1]
9
```

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

{ Perhatikanlah bahwa konstruktor pohon N-er dengan basis pohon kosong dituliskan sebagai

a. Prefix : (A,P,N)

b. Posfix : (PN,A) }

DEFINISI DAN SPESIFIKASI PREDIKAT

IsTreeNEmpty : PohonN-er \rightarrow boolean

{ IsTreeNEmpty(PN) true jika PN kosong : () }

IsOneElmt : PohonN-er \rightarrow boolean

{ IsOneElmt(PN) true jika PN hanya terdiri dari Akar }



```
1 def IsTreeEmpty(P):  
2     return P == []  
3  
4 def IsOneElmt (P):  
5     if not (IsTreeEmpty(P)) and IsTreeEmpty(Anak(P)):  
6         return True  
7     else :  
8         return False
```



DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

NbNElmt : PohonN-ner \rightarrow integer ≥ 0

{NbNElmt(P) memberikan banyaknya node dari pohon P :

Basis 1: NbNElmt ((A)\) = 1

Rekurens : NbNElmt ((A,PN)) = 1 + NbELmt(PN) }

NbNDaun : PohonN-ner \rightarrow integer ≥ 0

{NbNDaun (P) memberikan banyaknya daun dari pohon P :

Basis 1: NbNDaun (A) = 1

Rekurens : NbDaun ((A,PN)) = NbNDaun(PN)



Pohon Biner

Definisi : sebuah pohon biner adalah himpunan terbatas yang

- mungkin kosong, atau
- terdiri dari 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



Definisi rekursif pohon biner basis-0

- Basis : pohon biner kosong adalah pohon biner
- Rekurens : Pohon biner yang tidak kosong, terdiri dari sebuah node yang disebut akar, dan sub pohon kiri dan sub pohon kanan sebagai anak-anaknya yang juga merupakan pohon biner.

TYPE POHON BINER: Model -0, dengan basis pohon kosong

DEFINISI DAN SPESIFIKASI TYPE

type Elemen : { tergantung type node }

type PohonBiner : $\langle L : \text{PohonBiner}, A : \text{Elemen}, R : \text{PohonBiner} \rangle$ {notasi Infix}, atau

type PohonBiner : $\langle A : \text{Elemen}, L : \text{PohonBiner}, R : \text{PohonBiner} \rangle$ {notasi prefix}, atau

type PohonBiner : $\langle L : \text{PohonBiner}, R : \text{PohonBiner}, A : \text{Elemen} \rangle$ {notasi postfix }

{Pohon Biner terdiri dari Akar yang berupa elemen, L dan R adalah Pohon biner yang merupakan subPohon kiri dan subpohon kanan }

DEFINISI DAN SPESIFIKASI SELEKTOR

Akar : PohonBiner tidak kosong \rightarrow Elemen

{ Akar(P) adalah Akar dari P. Jika P adalah $\langle L, A, R \rangle = \text{Akar}(P)$ adalah A }

Left : PohonBiner tidak kosong \rightarrow PohonBiner

{ Left(P) adalah sub pohon kiri dari P. Jika P adalah $\langle L, A, R \rangle = \text{Left}(P)$ adalah L }

Right : PohonBiner tidak kosong \rightarrow PohonBiner

{ Right(P) adalah sub pohon kanan dari P. Jika P adalah $\langle L, A, R \rangle = \text{Right}(P)$ adalah R }

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

{Perhatikanlah bahwa konstruktor pohon biner dengan basis pohon kosong dituliskan sebagai

a. Infix : $\langle L A R \rangle$

b. Prefix : $\langle A L R \rangle$

c. Posfix : $\langle L R A \rangle$ }


DEFINISI DAN SPESIFIKASI PREDIKAT

IsEmpty : PohonBiner \rightarrow boolean

{IsEmpty (P) true jika P adalah Pohon biner kosong : $\langle \rangle$ }

DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

Notasi Prefix



```
1  def MakePB(A, L, R):
2      return [A, L, R]
3
4  def Akar(P):
5      return P[0]
6
7  def Left(P):
8      return P[1]
9
10 def Right(P):
11     return P[2]
```


NbElmt : PohonBiner \rightarrow integer ≥ 0

{NbElmt(P) memberikan Banyaknya elemen dari pohon P :

Basis : NbElmt ($// \wedge \backslash$) = 0

Rekurens : NbElmt ($/L,A,R\backslash$) = NbElmt(L) + 1 + NbElmt(R) }

NbDaun : PohonBiner \rightarrow integer ≥ 0

{ definisi : Pohon kosong berdaun 0 }

{NbDaun (P) memberikan Banyaknya daun dari pohon P :

Basis-1 : NbDaun ($// \wedge \backslash$) = 0

Rekurens :

NbDaun1 (P)

RepPrefix: PohonBiner \rightarrow list of element

{RepPrefix (P) memberikan representasi linier (dalam bentuk list), dengan urutan elemen list sesuai dengan urutan penulisan pohon secara prefix :

Basis : RepPrefix ($// \wedge \backslash$) = []

Rekurens : RepPrefix ($/L,A,R\backslash$) = [A] o RepPrefix(L) o RepPrefix (R) }

REALISASI

NbElmt (P) : {boleh model basis-0 }

if IsTreeEmpty?(P) then {Basis 0} 0

else {Rekurens } NbElmt (Left(P) + 1 + NbElmt(Right(P)

NbDaun (P) :

if IsEMpty?(P) then 0

else {Pohon tidak kosong:minimal mempunyai satu akar, sekaligus daun}

{ aplikasi terhadap Jumlah Daun untuk Basis-1 }

NbDaun1 (P)

RepPrefix (P) :

if IsTreeEmpty(P) then {Basis 0} []

else {Rekurens }

KonsoL(KonsoL(Akar(P), RepPrefix(Left(P)), RepPrefix(Right(P)))

```
1 def IsTreeEmpty(P):
2     if P == []:
3         return True
4     else :
5         return False
6
7 def NbElmt(P):
8     if IsTreeEmpty(P):
9         return 0
10    else :
11        return NbElmt(Left(P)) + 1 + NbElmt(Right(P))
12
13 def NbDaunPB(P):
14     if(IsTreeEmpty(P)) :
15         return []
16    else :
17        if IsOneElmtPB(P):
18            return 1
19        elif IsBinerPB(P):
20            return NbDaunPB(Left(P)) + NbDaunPB(Right(P))
21        elif IsUnerLeftPB(P):
22            return NbDaunPB(Left(P))
23        elif IsUnerRightPB(P):
24            return NbDaunPB(Right(P))
```

```
1 def RefPrefix(P):
2     if(IsTreeEmpty(P)) :
3         return []
4     else:
5         return KonsoL(KonsoL(Akar(P),RepPrefix(Left(P))),RepPrefix(Right(P)))
```



Definisi rekursif pohon biner basis-1

- Basis : pohon biner yang hanya terdiri dari akar
- Rekurens : Pohon biner yang tidak kosong, terdiri dari sebuah node yang disebut akar, dan sub pohon kiri dan sub pohon kanan sebagai anak-anaknya yang juga merupakan pohon biner tidak kosong

Notasi Prefix

TYPE POHON BINER : Model-1: pohon minimal mempunyai satu elemen

DEFINISI DAN SPESIFIKASI TYPE

type Elemen : { tergantung type node }

typ} PohonBiner : $\langle L : \text{PohonBiner}, A : \text{Elemen}, R : \text{PohonBiner} \rangle$ {notasi Infix}, atau

type PohonBiner : $\langle A : \text{Elemen}, L : \text{PohonBiner}, R : \text{PohonBiner} \rangle$ {notasi prefix },
atau

type PohonBiner : $\langle L : \text{PohonBiner}, R : \text{PohonBiner}, A : \text{Elemen} \rangle$ {notasi postfix }

{Pohon Biner terdiri dari Akar yang berupa elemen, L dan R adalah Pohon biner yang merupakan subPOhon kiri dan subpohon kanan }

DEFINISI DAN SPESIFIKASI SELEKTOR

Akar : PohonBiner tidak kosong \rightarrow Elemen

{ Akar(P) adalah Akar dari P. Jika P adalah //L A R\\ = Akar(P) adalah A }

Left : PohonBiner tidak kosong \rightarrow PohonBiner

{ Left(P) adalah sub pohon kiri dari P. Jika P adalah //L A R\\, Left (P) adalah L }

Right : PohonBiner tidak kosong \rightarrow PohonBiner

{Right(P) adalah sub pohon kanan dari P. Jika P adalah //L A R\\,Right (P) adalah R}

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

{Perhatikanlah bahwa konstruktor pohon biner dengan basis pohon kosong dituliskan sebagai

a. Infix : //L A R\\

b. Prefix : //A L R\\

c. Posfix : //L R A\\

atau bahkan notasi lain yang dipilih}



```
1  def MakePB(A, L, R):
2      return [A, L, R]
3
4  def Akar(P):
5      return P[0]
6
7  def Left(P):
8      return P[1]
9
10 def Right(P):
11     return P[2]
```

DEFINISI DAN SPESIFIKASI PREDIKAT

IsTreeEmpty : PohonBiner \rightarrow boolean

{IsTreeEmpty(P) true jika P kosong : ($// \backslash$) }

IsOneElmt : PohonBiner \rightarrow boolean

{IsOneElement(P) true jika P hanya mempunyai satu elemen, yaitu akar ($// A \backslash$) }

```
1 def IsTreeEmpty(P):
2     if P == []:
3         return True
4     else:
5         return False
6
7 def IsOneElement(P):
8     if not (IsTreeEmpty(P)) and IsTreeEmpty(Left(P)) and IsTreeEmpty(Right(P)):
9         return True
10    else:
11        return False
```


IsUnerLeft : PohonBiner \rightarrow boolean

{IsUnerLeft(P) true jika P hanya mengandung sub pohon kiri tidak kosong: (//L A \\\) }

IsUnerRight : PohonBiner \rightarrow boolean

{IsUnerRight (P) true jika P hanya mengandung sub pohon kanan tidak kosong: (//A R\\) }

IsBiner : PohonBiner tidak kosong \rightarrow boolean

*{IsBiner(P) true jika P mengandung sub pohon kiri dan sub pohon kanan :
(//L A R\\) }*



```
1 def IsUnerLeft(P):
2     if not IsTreeEmpty(P) and not IsTreeEmpty(Left(P)) and IsTreeEmpty(Right(P)):
3         return True
4     else:
5         return False
6
7 def IsUnerRight(P):
8     if not IsTreeEmpty(P) and IsTreeEmpty(Left(P)) and not IsTreeEmpty(Right(P)):
9         return True
10    else:
11        return False
12
13 def IsBiner(P):
14     if not IsTreeEmpty(P) and not IsTreeEmpty(Left(P)) and not IsTreeEmpty(Right(P)):
15         return True
16    else:
17        return False
```

IsExistLeft : PohonBiner tidak kosong \rightarrow boolean
{IsExistLeft (P) true jika P mengandung sub pohon kiri }

IsExistRight : PohonBiner tidak kosong \rightarrow boolean
{ExistRight(P) true jika P mengandung sub pohon kanan }

```
1 def IsExistLeftPB(P):
2     if (not IsTreeEmpty(P) and not IsTreeEmpty(Left(P))):
3         return True
4     else :
5         return False
6
7 def IsExistRightPB(P):
8     if (not IsTreeEmpty(P) and not IsTreeEmpty(Right(P))):
9         return True
10    else :
11        return False
```


DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

NbElmt : PohonBiner \rightarrow integer ≥ 0

{NbElmt(P) memberikan Banyaknya elemen dari pohon P :

Basis : NbElmt (//A\ \) = 1

Rekurens : NbElmt (//L,A,R\ \) = NbElmt(L) + 1 + NbELmt(R)

NbElmt (//L,A,\ \) = NbElmt(L) + 1

NbElmt (//A,R\ \) = 1 + NbELmt(R) }

NbDaun1 : PohonBiner \rightarrow integer ≥ 1

{ Prekondisi : Pohon P tidak kosong }

{NbDaun (P) memberikan Banyaknya daun dari pohon P :

Basis : NbDaun1 (//A\ \) = 1

Rekurens : NbDaun1 (//L,A,R\ \) = NbDaun1 (L) + NbDaun1(R)

NbDaun1 (//L,A,\ \) = NbDaun1 (L)

NbDaun1 (//A,R\ \) = NbDaun1 (R)

RepPrefix: PohonBiner \rightarrow list of element

{RepPrefix (P) memberikan representasi linier (dalam bentuk list), dengan urutan elemen list sesuai dengan urutan penulisan pohon secara prefix :

Basis : RepPrefix (//A\ \) = [A]

Rekurens : RepPrefix (//L,A,R\ \) = [A] o RepPrefix(L) o RepPrefix (R)

RepPrefix (//L,A,\ \) = [A] o RepPrefix(L)

RepPrefix (//A,R\ \) = [A] o RepPrefix (R)

```
1 def NBElement(P):
2     if IsOneElement(P):
3         return 1
4     else:
5         if (IsBiner(P)):
6             return NBElement(Left(P)) + 1 + NBElement(Right(P))
7         elif (IsUnerLeft(P)):
8             return NBElement(Left(P)) + 1
9         elif (IsUnerRight(P)):
10            return 1 + NBElement(Right(P))
11
12 def NBDaun(P):
13     if IsOneElement(P):
14         return 1
15     else:
16         if (IsBiner(P)):
17             return NBDaun(Left(P)) + NBDaun(Right(P))
18         elif (IsUnerLeft(P)):
19             return NBDaun(Left(P))
20         elif (IsUnerRight(P)):
21             return NBDaun(Right(P))
```

```
1 def RefPrefix(P):
2     if(IsTreeEmpty(P)) :
3         return []
4     else :
5         if IsOneElement(P):
6             return [Akar(P)]
7         else:
8             if (IsBiner(P)):
9                 return [Akar(P)] + RefPrefix(Left(P)) + RefPrefix(Right(P))
10            elif (IsUnerLeft(P)):
11                return [Akar(P)] + [RefPrefix(Left(P))]
12            elif (IsUnerRight(P)):
13                return [Akar(P)] + [RefPrefix(Right(P))]
```

Binary Search Tree

Definisi Binary Search Tree dengan key yang unik : Jika $P = /L A R\backslash$ adalah sebuah binary tree, maka:

- semua key dari node yang merupakan anak kiri P nilainya lebih kecil dari A, dan
- semua key dari node yang merupakan anak kanan P nilainya lebih besar dari A,

Definisi dan spesifikasi operasi terhadap binary search tree diberikan sebagai berikut. Realisasi nya harus dibuat sebagai latihan.

BSearchX : BinSearchTree, elemen \rightarrow boolean

{ BsearchX(P,X) Mengirimkan true jika ada node dari Pohon Binary Search Tree P yang bernilai X, mengirimkan false jika tidak ada }

AddX: BinSearchTree, elemen \rightarrow PohonBiner

{ AddX(P,X) Menghasilkan sebuah pohon Binary Search Tree P dengan tambahan simpul X. Belum ada simpul P yang bernilai X }

MakeBinSearchTree: list of elemen \rightarrow PohonBiner

{ MakeBinSearchTree(Ls) Menghasilkan sebuah pohon Binary Search Tree P yang elemennya berasal dari elemen list Ls yang dijamin unik. }

DelBtree: BinSearchTree tidak kosong, elemen \rightarrow PohonBiner

{ DelBTree(P,X) menghasilkan sebuah pohon binary search P tanpa node yang bernilai X. X pasti ada sebagai salah satu node Binary Search Tree. Menghasilkan Binary SearchTree yang "kosong" jika P hanya terdiri dari X }



Pohon Seimbang (balanced tree)

- Pohon seimbang tingginya: perbedaan tinggi sub pohon kiri dengan sub pohon kanan maksimum 1
- Pohon seimbang banyaknya simpul: perbedaan banyaknya simpul sub pohon kiri dengan sub pohon kanan maksimum 1



TERIMA KASIH

