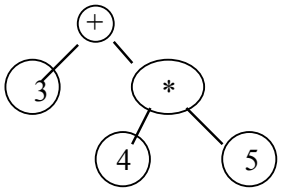
Pohon Biner

Definisi :

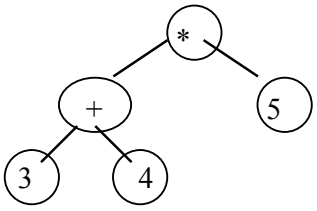
sebuah pohon biner adalah himpunan terbatas yang
- mungkin kosong, atau
- terdiri dari sebuah simpul yang disebut akar
  dan dua buah himpunan lain yang *disjoint* yang merupakan **pohon biner**, yang disebut sebagai sub pohon kiri dan sub pohon kanan dari pohon biner tersebut

Perhatikanlah perbedaan pohon biner dengan pohon biasa : pohon biner mungkin kosong, sedangkan pohon n-aire tidak mungkin kosong.

Contoh pohon ekspresi aritmatika

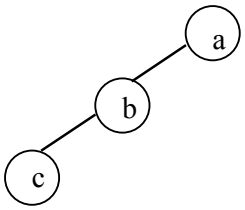


3+ (4*5)

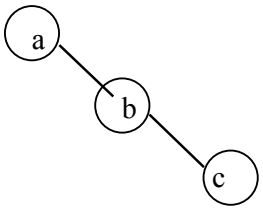


(3+4) * 5

Karena adanya arti bagi sub pohon kiri dan sub pohon kanan, maka dua buah pohon biner sebagai berikut berbeda (pohon berikut disebut pohon condong/skewed tree)

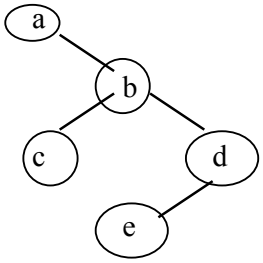


Pohon biner condong kiri



Pohon biner condong kanan

Sub pohon ditunjukkan dengan penulisan ( )



Notasi prefix :

(a ( ), (b (c () ())(d (e) ()))), atau
(a() (b (c) (d (e) ())))

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.

Jika pada list hanya ada dua cara melakukan konstruksi/seleksi yaitu pertama atau terakhir (perhatikan kata terdiri dari …), maka pada pohon biner tiga alternatif berikut dapat dipilih yaitu infix, prefix dan postfix. Pemilihan salah satu cara untuk implementasi disesuaikan dengan bahasanya. Contohnya karena dalam LISP ekspresi ditulis prefix, maka akan lebih mudah kalau dipilih secara prefix.

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

DEFINISI DAN SPESIFIKASI TYPE

type Elemen : { tergantung type node }

type PohonBiner : <L : PohonBiner, A : Elemen, R : PohonBiner > {notasi Infix}, atau

type PohonBiner : < A : Elemen, L : PohonBiner, R : PohonBiner > {notasi prefix}, atau

type PohonBiner : <L : PohonBiner, R : PohonBiner, A : Elemen > {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 → Elemen

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

Left : PohonBiner tidak kosong → PohonBiner

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

Right : PohonBiner tidak kosong → 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\ }

DEFINISI DAN SPESIFIKASI PREDIKAT

IsTreeEmpty : PohonBiner → boolean

{IsTreeEmpty (P) true jika P adalah Pohon biner kosong : (/ \) }

DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

NbElmt : PohonBiner → integer ≥ 0

{NbElmt(P) memberikan Banyaknya elemen dari pohon P :
Basis : NbElmt (//\ \) = 0
Rekurens : NbElmt (/L,A,R\) = NbElmt(L) + 1 + NbELmt(R) }

NbDaun : PohonBiner → integer ≥ 0

{ definisi : Pohon kosong berdaun 0 }
{NbDaun (P) memberikan Banyaknya daun dari pohon P :
Basis-1 : NbDaun (//\ \) = 0
Rekurens :
***NbDaun1** (P)*

RepPrefix: PohonBiner → list of element

{RepPrefix (P) memberikan representasi linier (dalam bentuk list), dengan urutan elemen list sesuai dengan urutan penulisan pohon secara prefix :
Basis : RepPrefix (//\ \) = []
Rekurens :RepPrefix (/L,A,R\) = [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)))
```

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

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

DEFINISI DAN SPESIFIKASI TYPE

type Elemen : { tergantung type node }

typ} PohonBiner : <L : PohonBiner, A : Elemen, R : PohonBiner > {notasi Infix}, atau

type PohonBiner : < A : Elemen, L : PohonBiner, R : PohonBiner > {notasi prefix }, atau

type PohonBiner : <L : PohonBiner, R : PohonBiner, A : Elemen > {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 → Elemen

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

Left : PohonBiner tidak kosong → PohonBiner

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

Right : PohonBiner tidak kosong → 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}

DEFINISI DAN SPESIFIKASI PREDIKAT

IsTreeEmpty : PohonBiner → boolean

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

IsOneElmt : PohonBiner → boolean

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

IsUnerLeft : PohonBiner → boolean

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

IsUnerRight : PohonBiner → boolean

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

IsBiner : PohonBiner tidak kosong → boolean

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

IsExistLeft : PohonBiner tidak kosong → boolean

{IsExistLeft (P) true jika P mengandung sub pohon kiri }

IsExistRight : PohonBiner tidak kosong → boolean

{ExistRight(P) true jika P mengandung sub pohon kanan }

DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

NbElmt : PohonBiner → 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 → 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 → 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)
}

REALISASI

```
NbElmt (P) : {P tidak kosong }
   if IsOneElmt(P) then 1
   else depend on P
            IsBiner(P) : NbElmt(Left(P) + 1 + NbElmt(Right(P)
            IsUnerLeft(P) : NbElmt(Left(P) + 1
```

```
            IsUnerRight(P) : 1 + NbElmt(Right(P)

NbDaun (P) :
   if 1Element?(P) then {Basis}
        1
   else {Rekurens }
         depend on P
            IsBiner(P) : NbDaun1(Left(P)) +  NbDaun1(Right(P))
            IsUnerLeft(P) : NbDaun1(Left(P))
            IsUnerRight(P) : NbDaun1(Right(P))

RepPrefix (P) :
   if 1Elmt?(P) then [Akar(P)]
   else depend on P
            IsBiner(P):KonsoL(KonsoL(Akar(P),RepPrefix(Left(P)),
                      RepPrefix(Right(P)))
            IsUnerLeft(P) : KonsoL(Akar(P),RepPrefix(Left(P))
            IsUnerRight(P) : KonsoL(Akar(P),RepPrefix(Right(P))
```

Latihan soal :

1. Pelajarilah apakah semua predikat pada model 1 berlaku untuk model 0
2. Buatlah spesifikasi dan definisi dari sebuah fungsi yang memeriksa apakah suatu nilai X ada sebagai simpul sebuah pohon
3. Buatlah spesifikasi dan definisi dari sebuah fungsi yang memeriksa apakah suatu pohon biner adalah sebuah pohon biner condong kiri
4. Buatlah spesifikasi dan definisi dari sebuah fungsi yang memeriksa apakah suatupohon biner adalah sebuah pohon biner condong kanan
5. Buatlah spesifikasi dan definisi dari sebuah fungsi yang menghitung banyaknya operator uner pada sebuah pohon ekspresi aritmatika infix
6. Buatlah spesifikasi dan definisi dari sebuah fungsi yang memeriksa banyaknya simpul yang ada pada level n
7. Latihan :

Buatlah realisasi dari spesifikasi fungsional terhadap pohon biner sebagai berikut

DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

IsMember : PohonBiner, elemen → boolean

{IsMember(P,X) Mengirimkan true jika ada node dari P yg bernilai X }

{ fungsi lain }

IsSkewLeft: PohonBiner → boolean

{ IsSkewLeft(P) Mengirimkan true jika P adalah pohon condong kiri }

IsSkewRight : PohonBiner → boolean

{ IsSkewRight(P) Mengirimkan true jika P adalah pohon condong kiri }

LevelOfX: PohonBiner, elemen → integer

{ LevelOfX(P,X) Mengirimkan level dari node X yang merupakan salah satu simpul dari pohon biner P }

{Operasi lain }

AddDaunTerkiri : PohonBiner, elemen → PohonBiner

{AddDaunTerkiri(P,X): mengirimkan Pohon Biner P yang telah bertambah simpulnya, dengan X sebagai simpul daun terkiri }

AddDaun : PohonBiner tidak kosong, node,node, boolean → PohonBiner

{AddDaun (P,X,Y,Kiri) : P bertambah simpulnya, dengan Y sebagai anak kiri X (jika Kiri), atau sebagai anak Kanan X (jika not Kiri)
{ Prekondisi : X adalah salah satu daun Pohon Biner P }

DelDaunTerkiri: PohonBiner tidak kosong, → <PohonBiner,elemen >

{DelDaunTerkiri(P) menghasilkan sebuah pohon yang dihapus daun terkirinya, dengan X adalah info yang semula disimpan pada daun terkiri yang dihapus }

DelDaun : PohonBiner tidak kosong, elemen → PohonBiner

{ DelDaun(P,X) dengan X adalah salah satu daun , menghasilkan sebuah pohon tanpa X yang semula adalah daun dari P}

MakeListDaun : PohonBiner →list Of Node

{MakeListDaun(P) : }
{Jika P adalah pohon kosong, maka menghasilkan list kosong.
{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua daun pohon P}

MakeListPreOrder : PohonBiner) →list Of Node

{MakeListPreOrder(P) : }
{Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua node pohon P dengan urutan Preorder}

MakeListPostOrder : PohonBiner →list Of Node

{MakeListPostOrder(P) : }
{Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua node pohon P dengan urutan PostOrder}

MakeListInOrder : PohonBiner →list Of Node

{MakeListInOrder(P) : }
{Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua node pohon P dengan urutan InOrder}

MakeListLevel : PohonBiner, integer →list Of Node

{MakeListLevel(P,N) : }
{Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua node pohon P yang levelnya=N}

Ada pohon biner yang mempunyai sifat-sifat khusus, misalnya pohon biner pencarian (binary search tree) dan pohon seimbang. Selain semua operator dan fungsi yang berlaku untuk pohon biner, ada operator lain yang didefinisikan.

Binary Search Tree

Definisi Binary Search Tree dengan key yang unik : Jika P = /L A R\ 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 → 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 → 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 → 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 → 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*)

Definisi Pohon seimbang:

- 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

Buatlah realisasi dari definisi dan spesifikasi sebagai berikut sebagai latihan

BuildBalanceTree: list of node, integer → BinBalTree *{ Meghasilkan sebuah balance tree dengan n node, nilai setiap node yang berasal dari list }*