MODUL X

Struktur Data

Judul	Struktur Pohon Dengan Linked List	
Penyusun	**Distribusi**	**Perkuliahan**
Nixon Erzed	Teknik Informatika Universitas Esa Unggul	Pertemuan – X online

Tujuan :

Mahasiswa memahami struktur pohon dan representasi dinamik pohon dengan linked list

Materi :

1. Pohon sebagai sebuah Graf
2. Pengertian Pohon
3. Jenis-jenis Pohon
4. Penelusuran (Traversal) Pohon
5. Pohon Biner
6. Traverse BTree
7. Operasi-operasi pada BTree
8. Representasi Pohon dengan Linked List

POHON

POHON SEBAGAI GRAF

Definisi Dasar :

Pohon adalah sebuah geraf (tidak berarah) terhubung yang tidak memiliki sirkuit. Dimana geraf G = (v,e) didefinisikan sebagai: himpunan objek yang disebut simpul (vertex) dan himpunan lain yang disebut sisi (edge), dimana sisi merepresentasikan hubungan antar objek.

Setiap sisi akan diidentifikasi oleh pasangan simpul (v_i , v_j).

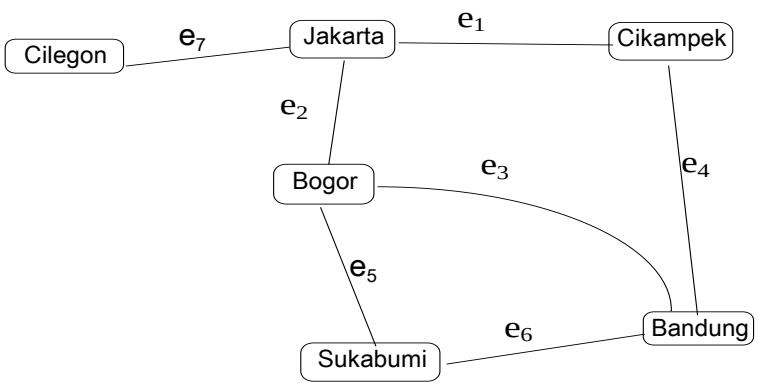
Contoh :

Himpunan kota : { Cilegon, Bandung, Jakarta, Cikampek, Bogor, Sukabumi }

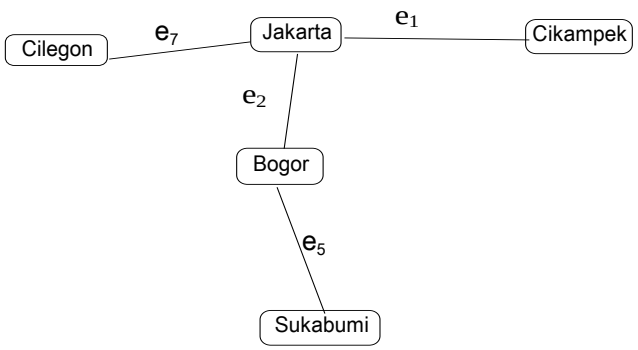
Himpunan sisi : { e_1 , e_2 , e_3 , e_4 , e_5 , e_6 , e_7 }, yang merepresentasikan jalan darat yang dapat ditempuh dari suatu kota ke kota lainnya.

Gambar (a) dan (b) berikut ini adalah representasi grafis dari Geraf G1 = (kota, lintasan) dan G2 = (kota, lintasan) :

(a)



(b)



Yang dimaksudkan sebagai geraf terhubung adalah jika untuk setiap pasang simpul selalu ada lintasan yang menghubungkannya. Jika pada geraf di atas ditambahkan sebuah kota lagi yaitu Pontianak, dimana tidak terdapat jalan darat yang menghubungkannya dengan kota lainnya, maka geraf tersebut menjadi tidak terhubung.

Sedangkan dari sifat tanpa sirkuit menyebabkan lintasan yang terdapat dalam geraf tersebut selalu lintasan tunggal, sederhana dan elementer. Dalam kalimat yang lebih sederhana hanya terdapat satu lintasan yang menghubungkan antara dua buah simpul. Pada contoh di atas, jika Bandung dihapus dari himpunan kota dan {e_3 ,e_4 , e_6} dihapus dari himpunan sisi, maka selalu ada hanya satu lintasan yang dapat menghubungkan dua buah simpul.

Geraf pada gambar diatas adalah sebuah *pohon* karena geraf tersebut terhubung dan tidak memiliki sirkuit

Geraf Berarah

Berdasarkan identifikasi terhadap sisi e_k, geraf dibedakan menjadi geraf-berarah dan geraf-tidak-berarah. Pada geraf-berarah pasangan (v_i , v_j) yang berasosiasi dengan sisi e_k, v_i adalah simpul asal dan v_j adalah simpul akhir, dan (v_i , v_j) $\neq$ (v_j , v_i). Sedangkan pada geraf tidak berarah (v_i , v_j) = (v_j , v_i).

Derajat Simpul

Jumlah sisi yang terhubung langsung pada simpul menyatakan derajat simpul tersebut. Pada geraf berarah, sebuah simpul memiliki dua macam derajat, yaitu derajat masuk (*in-degree*) yang menyatakan jumlah sisi yang masuk dan derajat-keluar (*out-degree*) yang menyatakan jumlah sisi keluar. Derajat sebuah simpul dalam geraf berarah merupakan jumlah derajat-masuk dan derajat-keluar.

Simpul berdarajat satu (1) disebut anting-anting (*pendant vertex*), dan simpul berderajat 0 disebut simpul terasing *(isolated vertex*).

Pohon Berakar

Sama seperti pengertian umum geraf, berdasarkan identifikasi sisi yang membangunnya, pohon dibedakan atas pohon tak berarah dan pohon berarah. Dalam pohon berarah, jika ada tepat satu simpul dengan derajat masuk 0 dan simpul lainnya memiliki derajat masuk 1, pohon tersebut dinamakan pohon berakar (directed-rooted-tree). Untuk pembahasan selanjutnya yang dimaksudkan adalah **pohon-berakar.**

Dikaitkan dengan implementasi pohon pada berbagai aplikasi, simpul merupakan elemen pohon yang mewakili item informasi, dan sisi merepresentasikan hubungan antara simpul-simpul atau kaitan antara item-item informasi tersebut.

Simpul dengan derajat-masuk = 0 disebut *simpul-akar* (selanjutnya disebut akar), simpul yang memiliki derajat-masuk = 1 dan derajat-keluar $\neq 0$ disebut simpul cabang (selanjutnya disebut cabang), sedangkan simpul dengan derajat masuk = 1 dan derajat keluar = 0. daun (selanjutnya disebut daun).

Level Pohon

Panjang lintasan (jumlah sisi) dari akar ke suatu simpul menujukkan tingkat (level) simpul tersebut.

Untuk pengertian yang berlaku umum, tingkat maksimun dari suatu pohon sama dengan panjang lintasan dari akar ke daun terjauh. Akar berada di tingkat 0 dan daun terjauh berada ditingkat maksimum.

Pohon m-Cabang

Pohon m-cabang (m-ary tree) adalah pohon yang simpulnya maksimum memiliki m anak. Pohon yang tiap cabang memiliki jumlah anak sama disebut pohon teratur (regular tree), jika jumlah anak tersebut adalah m maka disebut Pohon-Teratur m-Cabang (regular m-ary tree).

Secara umum Pohon-Teratur m-Cabang memiliki sifat sebagai berikut :

- mempunyai paling sedikit m sisi dan (m+1) simpul,
- mempunyai tepat satu simpul berderajat m,
- simpul lain berderajat 1 atau (m+1)

Dalam struktur data hirarki, pohon teratur m-Cabang yang populer diantaranya : Pohon Teratur 2-Cabang (pohon biner), dan Pohon Teratur 4-Cabang (Pohon Empatan).

PENGERTIAN POHON

Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hirarkis (hubungan one to many) antara elemen-elemen.

Tree bisa didefinisikan sebagai kumpulan simpul/node dengan satu elemen khusus yang disebut Root/Akar dan node lainnya terbagi menjadi himpunan himpunan yang saling tidak berhubungan satu sama lain (disebut sub-tree).

Untuk jelasnya dibawah akan diuraikan istilah-istilah umum dalam Tree :

a. Predeccessor	:	node yang berada diatas node tertentu.
b. Succerssor	:	node yang berada dibawah node tertentu.
c. Ancestor	:	seluruh node yang terletak sebelum node tertentu dan berada pada jalur (path) yang sama.
d. Descendant	:	seluruh node yang terletak setelah node tertentu dan berada pada jalur (path) yang sama.
e. Parent	:	predecessor satu level diatas suatu node
f. Child	:	successor satu level dibawah suatu node.
g. Sibling	:	node-node yang memiliki parent yang sama
h. Subtree	:	bagian dari tree yang berupa suatu node beserta descendantnya dan memiliki semua karakteristik dari tree tersebut.
i. Size	:	Banyaknya node dalam satu tree.
j. Heigh	:	Banyaknya tingkatan/level dalam satu tree.
k. Path	:	lintasan yang menghubungkan suatu node hingga Root /Akar
l. Leaf / DauN	:	node yang tidak memiliki successor
m. Forest	:	himpunan pohon-pohon

Jenis-jenis pohon :

Jenis pohon dibedakan berdasarkan jumlah anak (cabang) setiap nodenya. Penamaan pohon juga mengikuti pola tersebut.

a. Pohon 2 : pohon yang setiap nodenya memiliki anak paling banyak 2 dikenal juga sebagai Pohon Biner

b. Pohon 3 : pohon yang setiap nodenya memiliki anak paling banyak 3

c. Pohon 4 : pohon yang setiap nodenya memiliki anak paling banyak 4

d. Pohon n : pohon yang setiap nodenya memiliki anak paling banyak n

Sebuah pohon biner dapat juga terdiri dari node-node yang secara teratur selalu memiliki anak 2 atau tidak ada sama sekali, yang dikenal sebagai Pohon Biner Teratur (Regular Binary Tree).

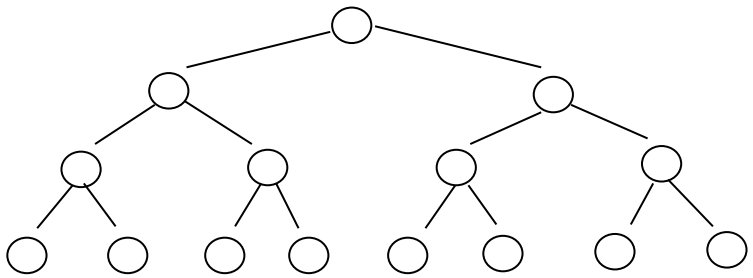
Secara terpola, untuk pohon teratur m cabang disebut Regular m-ary Tree.

Berdasarkan kompleksitasnya, pohon dibedakan atas :

a. Pohon-m Penuh

Pohon-m cabang yang tiap nodenya memiliki *m* atau 0 anak dan setiap leaf memiliki panjang path yang sama.

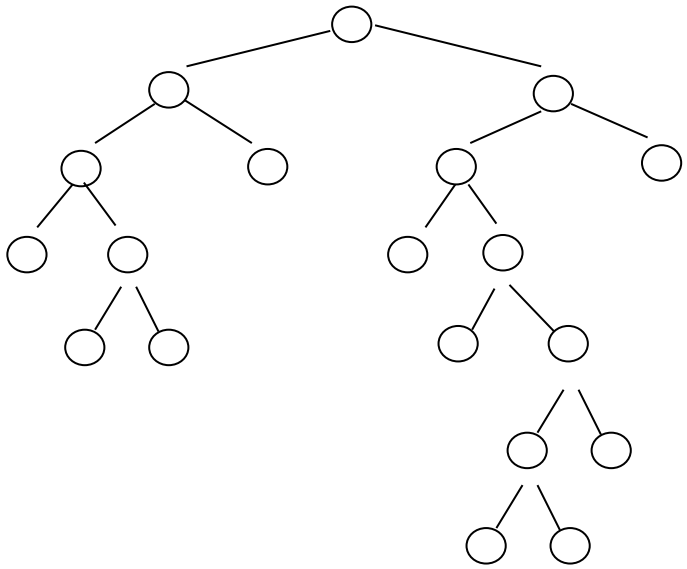
Contoh : *pohon biner penuh*



b. Pohon-m Lengkap atau Pohon Teratur

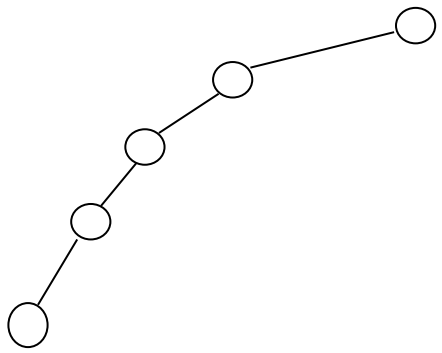
Pohon-m cabang yang tiap nodenya memiliki *m* atau 0 anak

Contoh : *pohon biner lengkap*



Pohon-m Skewed.

Pohon m cabang yang semua nodenya kecuali leaf, hanya memiliki 1 anak.

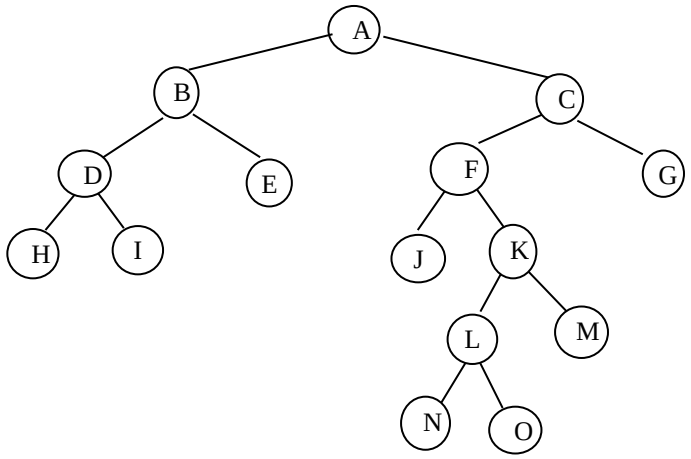


PENELUSURAN POHON (TRAVERSAL)

Untuk mengunjungi setiap simpul pohon, selalu dimulai dari Root. Terdapat dua teknik penelusuran (traversal), yaitu :

1. Breadth First Traversal (BFT)
   Penelusuran dimulai dari Root, selanjutnya anak paling kiri pada level 1 dan dilakukan secara horizontal, dan dilanjutkan ke level berikutnya.
2. Depth First Traversal (DFT)
   Penelusuran dimulai dari Root, selanjutnya anak paling kiri pada level 1, dan dilanjutkan hingga anak paling kiri terjauh. Jika ditemukan anak paling jauh pada cabang paling, lanjutkan pada siblingnya.

Jika diberikan pohon biner sebagai berikut :



Maka hasil penelusurannya adalah

BFT :

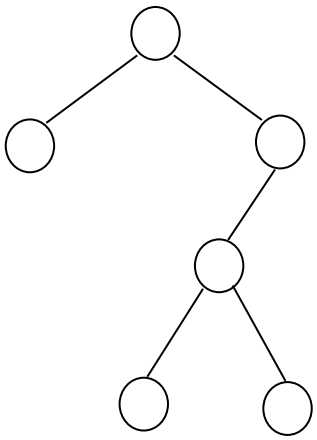
A B C D E F G H I J K L M N O

DFT :

A B D H I E C F J K L N O M G

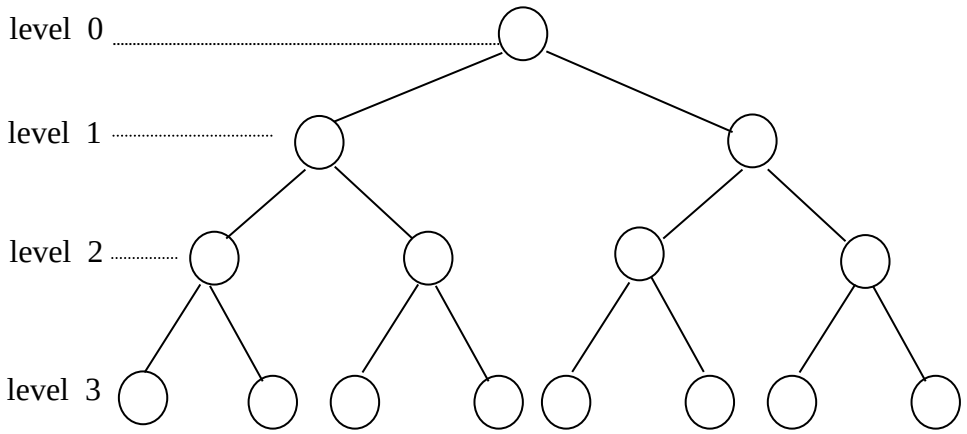
BINARY TREE

Binary Tree, adalah tree dengan node yang memiliki jumlah anak maksimum 2.

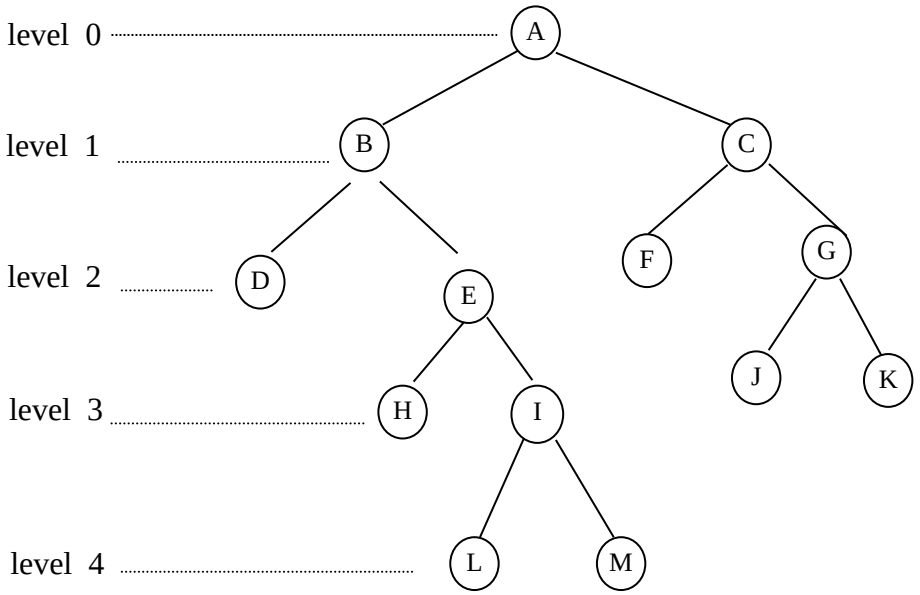


Semua sifat-sifat umum Tree, merupakan sifat Binary Tree. Berikut ini adalah bentuk-bentuk khusus Binary Tree.

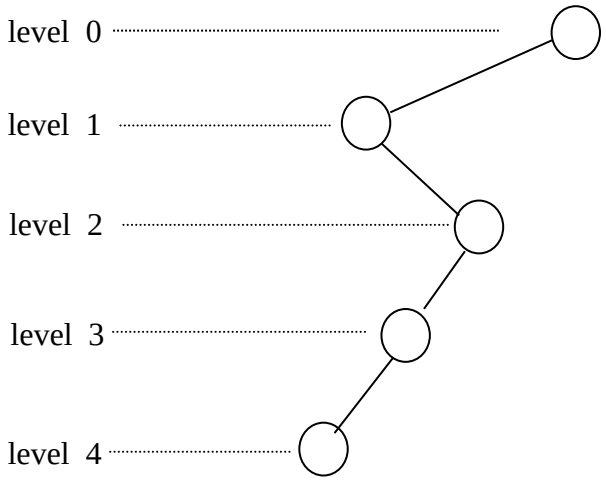
a. **Full Binary Tree** : Binary Tree yang tiap nodenya (kecuali leaf) memiliki dua child/anak dan setiap sub tree harus mempunyai panjang path yang sama (leaf berada pada level yang sama)



b. **Complete Binary Tree** : Binary Tree yang tiap nodenya (kecuali leaf) memiliki dua child/anak, tetapi leaf dapat berada pada level yang berbeda



c. **Skewed Binary Tree** : Binary Tree yang tiap nodenya (kecuali leaf) hanya memiliki satu child/anak.



TRAVERSE BTree

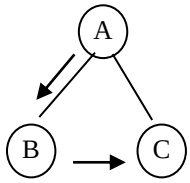
Traverse binary tree, adalah mengunjungi seluruh node-node pada tree, masing-masing satu kali. Hasilnya adalah urutan informasi secara linear yang tersimpan dalam tree. Terdapat tiga cara traverse :

- PreOrder
- InOrder
- PostOrder

Langkah-langkah traverse :

PreOrder

Visit isi node current, kunjungi LeftChild, kunjungi RightChild

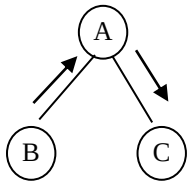


Untuk contoh Binary Tree yang diberikan pada Complete Binary Tree, hasil traverse secara Pre Order adalah :

A B D E H I L M C F G J K

InOrder

Kunjungi LeftChild, visit isi node current, kunjungi RightChild

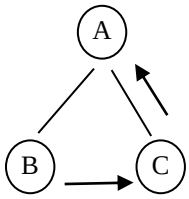


Untuk contoh Binary Tree yang diberikan pada Complete Binary Tree, hasil traverse secara In Order adalah :

D B H E L I M A F C J G K

Post Order

Kunjungi LeftChild, kunjungi RightChild, visit isi node current



Untuk contoh Binary Tree yang diberikan pada Complete Binary Tree, hasil traverse secara Post Order adalah :

D H L M I E B F J K G C A

Operasi-operasi pada Binary Tree :

1. Create

   Procedure Create berguna untuk menciptakan BTree baru dan Kosong.

2. Empty

   Function Empty adalah untuk memeriksa apakah Btree masih kosong.

3. Insert

   Adalah prosedur untuk menyisipkan node baru kedalam Tree. Terdapat 3 pilihan yang mungkin : insert sebagai root (jika dan hanya jika Btree Empty), sebagai Left Child/Anak Kiri, dan sebagai Right Child / Anak Kanan

4. Find

   Mencari data tertentu pada Tree, merupakan pemanfaatan Traverse

5. Update

   Mengubah isi dari node yang ditunjuk oleh Current (Current adalah variabel pointer yang dideklarasikan untuk digunakan dalam penelusuran pohon)

6. Retrieve

   Mengetahui isi dari node yang ditunjuk oleh current

7. DeleteSub

   Menghapus sebuah subtree yang ditunjuk oleh current (node beserta seluruh descentdantnya)

8. Clear

   Procedure Clear berguna untuk mengosongkan Btree yang sudah ada. Operasi pengosongan dilakukan dengan cara menghapus seluruh node yang dimulai dari node leaf. Untuk menghindarkan kerumitan logika, maka dilakukan pemanggilan rekursif terhadap Procedure DeleteSub.

REPRESENTASI POHON DENGAN LINKED LIST

Untuk implementasi struktur pohon dalam program, umumnya digunakan linked list. Jumlah field pointer pada linked list merepresentasikan jumlah maksimun cabang pada pohon, atau dengan kata lain implementasi *pohon-m Cabang* memerlukan deklarasi m field pointer.

Contoh :

Deklarasi data untuk pohon biner :

```
Type
    pSimpul = ^Simpul
    Simpul = record of
             Data : TipeData
             Left, Right : Tree
        End;
```

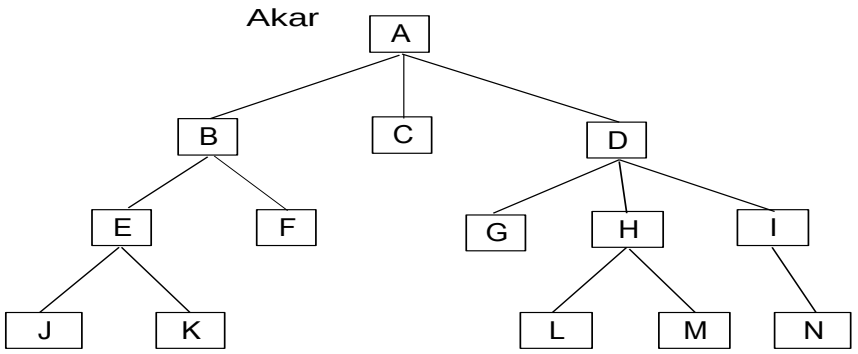
Deklarasi Data untuk Pohon 4 Cabang (*Quadtree*)

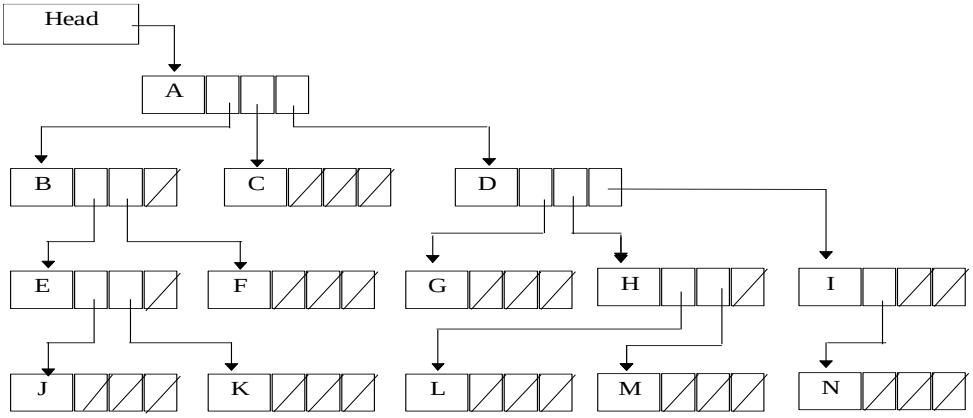
```
Type
     pSimpul = ^Simpul
    Node = record of
             Isi : TipeData
             Anak1, Anak2, Anak3, Anak4    : Tree
        End;
```

Deklarasi Data untuk Pohon ***n*** Cabang :

```
Type
    pSimpul = ^Simpul;
    Simpul  = record
              data                          :  <type data>;
              anak1, anak2, …, anak_n       : pSimpul
          end;
```

Contoh ilustrasi Tree yang disusun dengan Linked List :



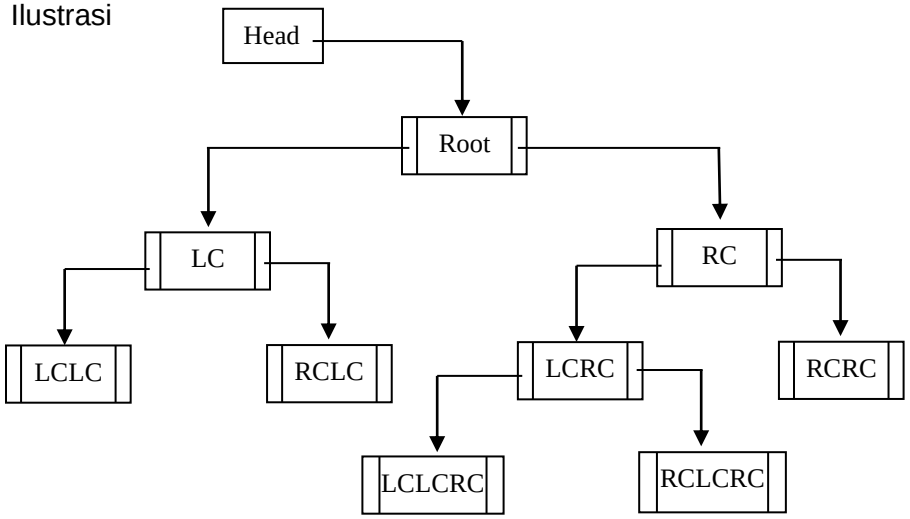


BTREE dengan Linked List

Implementasi Binary Tree dalam program umumnya menggunakan double linked list. Berikut ini adalah contoh deklarasi Linked List untuk merepresentasikan Binary Tree:

```
type
      BTree  = ^simpul
      Node   = record
               Data    : typedata;
               Kiri, Kanan :  Btree;
           end;
var
      head  :  Btree
```

Ilustrasi



LC : Left child (anak kiri)

RC : Right child (anak kanan)