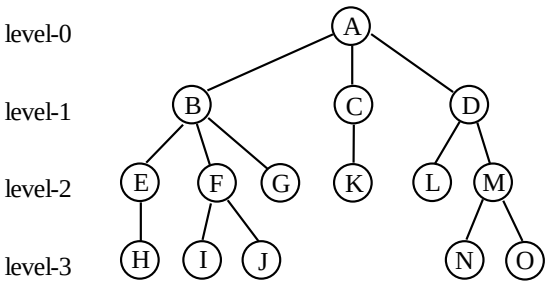
IV. STRUKTUR TREE (POHON)

4.1. KONSEP DASAR.

HIRARKI, LEVEL, ROOT (AKAR) dan LEAF (DAUN)

Struktur Tree (Pohon) adalah struktur yang mengandung aspek *hirarki*. Hirarki dibentuk melalui pengelompokkan *elemen* (disebut juga *node* atau *vertex*) dalam level level. Dimulai dari level 0 (biasanya digambar paling atas) diikuti oleh level 1, 2 dan seterusnya.
Hubungan hirarkis ada antara 2 level berurutan, biasa disebut sebagai hubungan *ATASAN-BAWAHAN*, *PARENT-CHILDREN*, *FATHER-SON*, atau *OWNER-MEMBER*. Sebuah node *BAWAHAN (CHILD/SON/MEMBER)* hanya boleh mempunyai satu *ATASAN (PARENT/FATHER/OWNER).* Akan tetapi satu node sebagai *ATASAN (PARENT/FATHER/OWNER)* boleh mempunyai lebih dari 1 *BAWAHAN (CHILD/SON/MEMBER)*, atau tidak sama sekali.
Hirarki dimulai dari level 0, yang hanya boleh ditempati oleh 1 buah node, disebut sebagai *ROOT (AKAR)*. Root boleh mempunyai beberapa *anak* (*bawahan/ children/ son/ member*), yang merupakan node yang berada pada level 1. Selanjutnya, setiap node pada level 1 ini boleh pula mempunyai beberapa anak pada level 2. Demikian seterusnya sehingga terbentuk suatu struktur tree.
Node yang tidak mempunyai anak disebut sebagai *LEAF* (jamaknya *LEAVES*, atau *DAUN*), atau *TERMINAL-NODE*. Sedangkan node yang mempunyai anak disebut sebagai *NON- TERMINAL-NODE*.
DEGREE suatu node ditentukan oleh jumlah anak node ybs., sedangkan *DEGREE* suatu tree adalah jumlah anak maksimum yang dimiliki oleh node pada tree ybs.
Sebagai ilustrasi : perhatikan gambar tree disamping ini.
maka :

- node A merupakan *ROOT* tree ini.
- node H, I, J, G, K, L, N dan O adalah *LEAVES*, atau *TERMINAL-NODE*.
- node A, B, C, D, E, F, L dan M adalah node *NON-TERMINAL*.
- node B, C dan D adalah *BROTHERS*, demikian pula
    - node E, F dan G
    - node I dan J
    - node L dan M
    - node N dan O



- node B, C dan D berada pada *level 1*.
- node E, F, G, K, L, dan M berada pada *level 2*.
- node H, I, J, N dan O berada pada *level 3*.
- *degree* dari setiap node :

NODE	DEGREE	NODE	DEGREE	NODE	DEGREE	NODE	DEGREE
A	3	E	1	I	0	M	2
B	3	F	2	J	0	N	0
C	1	G	0	K	0	O	0
D	2	H	0	L	0		

Sedangkan tree ini berdegree 3, karena degree maksimum nodenya (ialah node A atau B) = 3.

- B adalah *first-son* (anak pertama) dari A, sedangkan D adalah *last-son* dari A.
- C adalah *next-brother* dari B, demikian juga D terhadap C.
- Sedangkan B adalah *prior-brother* dari C, demikian juga C terhadap D.
- Sebutan yang sama dapat dibuat pada kumpulan lain seperti :
  B dan E, F, G
  F dan I, J dll.

Terlihat bahwa struktur tersebut dibayangkan sebagai pohon terbalik, dengan akar digambar diatas, batang, cabang dan daun "tumbuh" kearah bawah.
Contoh penggunaan : struktur organisasi, hirarki keluarga (silsilah), daftar isi buku, dll.

Sifat rekursif *tree* :

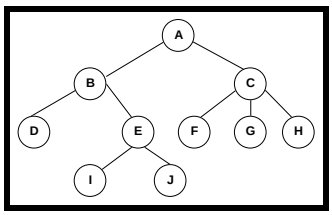
1. Sebuah simpul tunggal merupakan suatu pohon,

2. Bila terdapat simpul n dan beberapa sub pohon T1, T2,...,Tk yang tidak saling berhubungan, yang masing-masing akarnya n1,n2,n3, ..., nk.
   Dari simpul/sub pohon tersebut dapat dibuat sebuah pohon baru dengan n sebagai root dari simpul/sub pohon n1,n2,...,nk

Istilah dasar :

1. Simpul/node/vertex adalah elemen-elemen pohon yang mengandung informasi (data) menunjuk pencabangan.

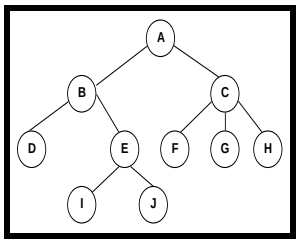
   

2. Akar/Root,
   a. Pada contoh di atas, rootnya adalah simpul A
   b. Hubungan/relasi antara simpul dianalogikan dengan hubungan keluarga/silsilah
   Contoh : -. A *parent/Owner* (orang tua/Atasan) dari simpul B dan C
   -. Sebaliknya B dan C merupakan *sons/ Member* (anak/bawahan) dari A.
   -. Simpul yang memiliki parent yang sama adalah *siblings/brothers* (bersaudara).
   Misalnya : B bersaudara dengan C
   D bersaudara dengan E
   I bersaudara dengan J
3. Tingkat/level dari simpul.
   a. Level dari *root* didefinisikan sebagai tingkat/level 0.
   b. Tingkat simpul lainnya didefinisikan sebagai 1+tingkat dari parent simpul tersebut.
   Dari contoh di atas :

Tingkat	Simpul-simpul
0	A
1	B,C
2	D,E,F,G,H
3	I,J

4. *Depth*/Kedalaman dari suatu pohon dinyatakan oleh tingkat yang tertinggi dari simpul yang terdapat pada pohon tersebut. Jadi suatu pohon dikatakan memiliki kedalaman n, bila paling sedikit ada satu simpul yang bertingkat n.
5. Derajat Simpul (*Order*)
   Sebuah pohon dikatakan berorder n, bila memiliki n tahap turunan menuju simpul terendah.

Tree dibawah memiliki depth = 3 (~ n)



Order	Simpul-simpul
0	I,J,D, F, G
1	E,C
2	B
3	A

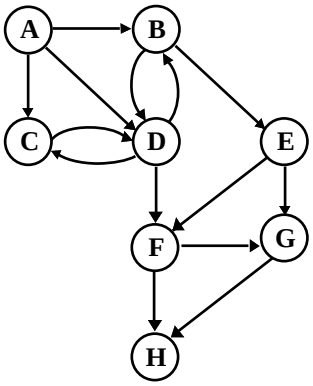
Simpul-simpul berderajat/order 0 (nol) adalah simpul yang tidak mempunyai turunan, disebut simpul terminal/ *daun/leaf*.
Simpul yang bukan berderajat 0 disebut simpul non terminal/internal.

6. *Degree* suatu Node ditentukan oleh jumlah anak Node ybs. Sedangkan Degree suatu Tree adalah jumlah anak maksimum yang dimiliki oleh node pada tree tsb.

STRUKTUR TREE bersifat "KONSEPTUAL"

Simaklah ilustrasi masalah sbb :

Misalnya ada jaringan jalan raya antar lokasi seperti diwakili oleh diagram seperti pada gambar. Perhatikan bahwa arah panah memperlihatkan arah lalu lintas pada jalur jalan ybs.
Kemudian diminta untuk menyusun semua kemungkinan perjalanan yang dapat dilakukan dari lokasi A ke lokasi H, dengan syarat bahwa pada suatu perjalanan tidak boleh ada lokasi yang dilalui 2 x.
Sebagai contoh misalnya :

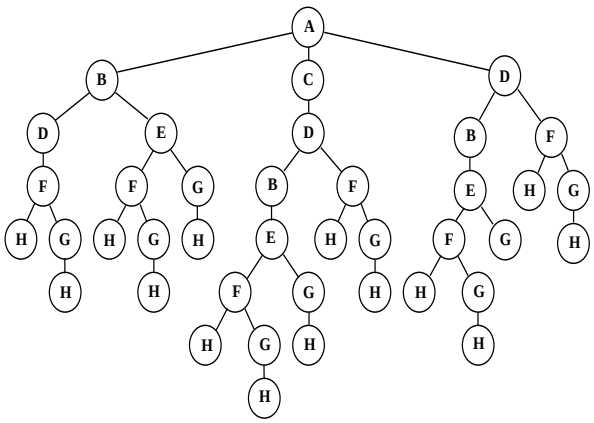


jalur *A-C-D-B-E-F-H* ataupun *A-B-D-F-H* adalah jalur yang sah.

sedangkan :

jalur *A-D-C-D-F-H* adalah contoh jalur tidak sah.

Semua kemungkinan jalur yang diminta tersebut dapat digambar sebagai sebuah tree (lihat gambar). Cara pembuatan tree tersebut diatas :



Dari lokasi A ada 3 lokasi yang terhubung ialah C, D dan B. Kemudian dilakukan penelusuran untuk ketiga alternatif sbb :

- dari C hanya ada jalur ke D
- dari D ada jalur ke C, B dan F
  - akan tetapi ke C tidak boleh,
  - sehingga tinggal ke B dan F
- dari B ada jalur ke D dan E

Demikian seterusnya untuk setiap jalur dilakukan penelusuran kemungkinan dengan memperhatikan boleh tidaknya jalur lanjutan tersebut ditempuh.

Sebagai hasil akhir akan diperoleh sebuah tree yang merupakan representasi dari semua kemungkinan perjalanan dari A ke H, root tree adalah A dan setiap leaf adalah H.
Dari contoh ini dapat disimpulkan bahwa pada struktur tree, penggunaan istilah seperti atasan/bawahan atau ayah/anak bersifat *abstrak*.

4.2. TRAVERSING.

Adalah proses ***visiting*** (mengunjungi) setiap node pada sebuah tree masing masing 1 x. Pada kunjungan ini tidak dipermasalahkan hal hal seperti : *tujuan* sebenarnya kunjungan tersebut, ataupun *jenis aksi* yang harus dilakukan pada setiap node yang sedang dikunjungi.
Traversing dapat dilakukan dengan menuruti suatu aturan urutan (*order*). Beberapa diantaranya misalnya : ***Pre-order***, ***In-order***, ***Post-order*** dan ***Level-order***.

TRAVERSING PREORDER

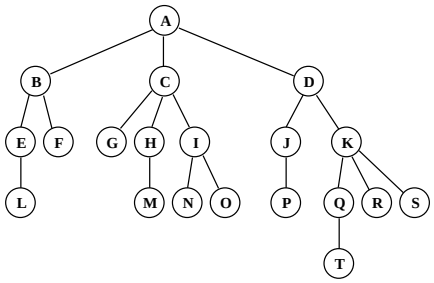
Aturan urutan kunjungan yang dipakai adalah :

- Kunjungi *Root* ,
- Kunjungi *SubTree ke 1,*
- Kunjungi *SubTree ke 2,*

.... dst jika ada

aturan diatas dilaksanakan secara terus menerus, sehingga aturan tersebut bersifat *rekursif*.

Contoh :
Perhatikan struktur tree sbb :



Penerapan aturan preorder pada level 0 dan 1 :

-kunjungi A,
-kunjungi subtree dengan "root" B
-kunjungi subtree dengan "root" C
-kunjungi subtree dengan "root" D

atau secara singkat ditulis :

A, *(B), (C), (D)*

tanda kurung menyatakan subtree dengan "root" node ybs !

selanjutnya penerapan aturan PREORDER pada subtree dengan "root" B :

B, *(E),* **F**

sedangkan pada subtree dengan "root" C :

C, G, *(H), (I)*

dan pada subtree dengan "root" D :

D, *(J), (K)*

Jika ketiganya disubstitusikan maka diperoleh hasil sementara :

A, B, *(E)***, F, C, G,** *(H)***,** *(I)***, D,** *(J)***,** *(K)*

Kemudian dilakukan penguraian traversing pre-order untuk subtree dengan "root" berturut turut E, H, I, J dan K sehingga hasil sementara menjadi :

A, B, E, L, F, C, G, H, M, I, N, O, D, J, P, K, *(Q)***, R, S**

Setelah traversing preorder subtree "Q" disubstitusikan, maka diperoleh hasil lengkap proses TRAVERSING PREORDER untuk seluruh tree sbb :

A, B, E, L, F, C, G, H, M, I, N, O, J, P, K, Q, T, R, S

TRAVERSING POSTORDER

dilakukan dengan aturan umum

- Kunjungi *SubTree ke 1*,
- Kunjungi *SubTree ke 2*,

  .... dst jika ada
- Kunjungi *Root* ,

perhatikan bahwa root/ayah dikunjungi terakhir, sesudah semua subtree/anak nya selesai dikunjungi.

Untuk bentuk tree seperti pada contoh preorder, maka pengembangan traversing postorder adalah sbb :

*(B)***,** *(C)***,** *(D)***, A**

Kemudian :

*(E)***, F, B, G,** *(H)***,** *(I)***, C,** *(J), (K)***, D, A**

Selanjutnya :

L, E, F, B, G, M, H, N, I, O, C, P, J, *(Q)***, R, S, K, D, A**

dan akhirnya :

E, L, F, B, G, M, H, N, O, I, C, P, J, T, Q, R, S, K, D, A

TRAVERSING INORDER

Ada 2 pendapat mengenai traversing jenis ini :

a. tidak terdefinisi untuk binary tree (Knuth).

b. terdefinisi dengan aturan sbb :

- Kunjungi *SubTree ke 1*,
- Kunjungi *Root* ,
- Kunjungi *SubTree ke 2*,

.... dst jika ada

Sesuai pendapat pada butir b, untuk bentuk tree seperti pada contoh preorder, maka pengembangan traversing inorder adalah sbb :

(B), **A** *, (C), (D)*

Kemudian :

*(E)***, B, F, A, G, C,** *(H), (I), (J)***, D,** *(K)*

Selanjutnya :

E, L, B, F, A, G, C, M, H, N, I, O, P, J, D, *(Q)***, K, R, S**

dan akhirnya :

E, L, B, F, A, G, C, M, H, N, I, O, P, J, D, T, Q, K, R, S

TRAVERSING LEVEL-ORDER

Dilakukan dengan aturan bahwa kunjungan dilakukan dengan mendahulukan setiap node pada level atas mulai dari kiri kekanan.
Untuk tree seperti pada contoh traversing preorder maka hasilnya adalah :

A,	**B,C,D,**	**E,F,G,H,I,J,K,**	**L,M,N,O,P,Q,R,S,**	**T**
level-0	level-1	level-2	level-3	level-4

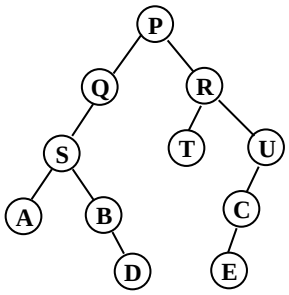
Cara seperti ini sering pula disebut sebagai traversing dengan strategi "*breadth-first*" (lebar didahulukan).
Secara prinsip merupakan lawan dari strategi "*depth-first*" (dalam didahulukan), yang merupakan sebutan lain untuk traversing Pre-order.

4.3. BINARY TREE dan FOREST

DEFINISI BINARY TREE

Binary tree termasuk keluarga tree yang penting, karena sering dipergunakan sebagai landasan pada berbagai penyelesaian masalah didunia informatika.
Sifat khusus sebuah binary tree adalah :

- degree maksimum setiap node adalah 2, dengan kata lain setiap node hanya boleh memiliki maksimum 2 anak.
- Anak pertama disebut *LEFT-SON* (*MEMBER/ CHILD*) sedangkan anak kedua disebut sebagai *RIGHT-SON*.
- dalam kasus dimana hanya ada 1 anak, diperkenankan untuk hanya mempunyai *LEFT-SON* atau *RIGHT-SON* saja.

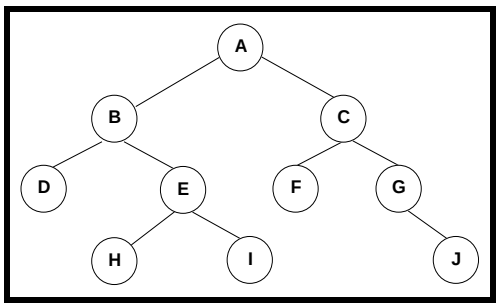


Sebagai ilustrasi, perhatikan binary tree dibawah ini :

- node Q dan C hanya mempunyai left-son
- node B dan T sebaliknya hanya mempunyai right-son

- Banyak simpul maksimum pada tingkat ke-n dari *binary tree* adalah (2^{n}).
- Banyak simpul maksimum dengan kedalaman n dari *binary tree* adalah ($2^{n+1} - 1$).

Contoh :



Tingkat	Banyak simpul Maksimal
0	$2^{n} = 2^{0} = 1$
1	$2^{n} = 2^{1} = 2$
2	$2^{n} = 2^{2} = 4$
3	$2^{n} = 2^{3} = 8$

Tree di atas memiliki depth = 3 (~ n), sehingga simpul maksimumnya adalah : ($2^{n+1} - 1$) = 15.

TRAVERSING INORDER

Aturan urutan kunjungannya (bersifat rekursif) adalah :

- Kunjungi *Subtree Kiri* ,
- Kunjungi *Root* ,
- Kunjungi *Subtree Kanan*

Maka, untuk binary tree seperti pada contoh diatas :
Penerapan pada level 0 dan 1 menghasilkan :

-kunjungi subtree kiri (dengan "root" Q)
-kunjungi node P
-kunjungi subtree kanan (dengan "root" R)

atau secara singkat :

(Q), **P,** *(R)*

selanjutnya "Q" dan "R" diuraikan sehingga diperoleh :

(S), **Q, P,** *(T)*, **R, U**

kemudian :

A, S, *(B)*, **Q, P, T,** *(C)*, **R, U**

dan akhirnya :

A, S, B, D, Q, P, T, E, C, R, U

Catatan :
Jangan lupa bahwa traversing Pre-order dan Post-order berlaku pula untuk binary tree. Untuk binary tree contoh ini, hasilnya adalah :

preorder **P, Q, S, A, B, D, R, T, C, E, U**
postorder **A, D, B, S, Q, E, C, T, U, R, P**

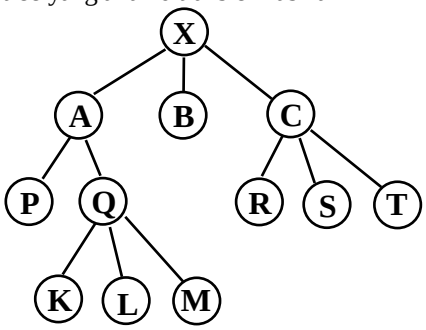
TRANSFORMASI TREE menjadi BINARY-TREE

Didunia informatika, pada umumnya binary tree jauh lebih disukai sebagai objek dari pada tree. Oleh karena itu sebuah tree bukan biner seringkali "*diubah*" sehingga menjadi binary-tree. Proses ini disebut sebagai "*transformasi*", dan dilaksanakan dengan cara sbb :

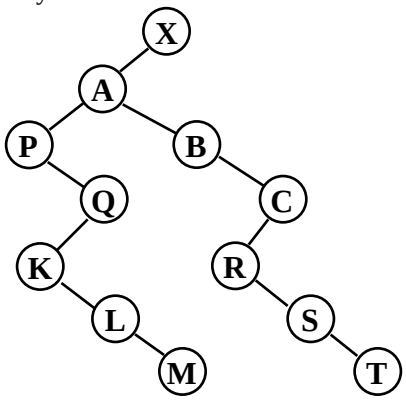
FIRST-SON pada tree dianggap sebagai *LEFT-SON*, dan
NEXT-BROTHER pada tree diangap sebagai *RIGHT-SON*.

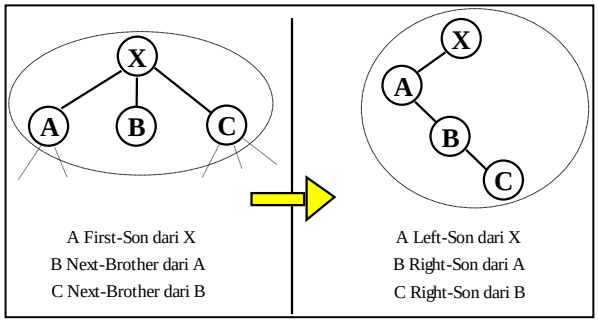
Contoh :

tree yang akan ditransformasikan



binary tree hasil transformasi





misalkan diketahui tree seperti pada gambar, maka transformasi menjadi binary tree dilakukan sbb :

- perhatikan level 0 dan 1 (X, A, B, C)

  - A adalah *First-Son* dari X
  - B adalah *Next-Brother* dari A
  - C adalah *Next-Brother* dari C

- jika ditransformasikan maka menjadi :
  - A adalah *Left-Son* dari X
  - B adalah *Right-Son* dari A
  - C adalah *Right-Son* dari C

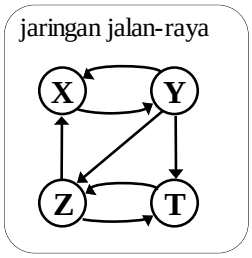
demikian seterusnya dilakukan untuk node lainnya, sehingga akan diperoleh hasil seperti pada gambar.

FOREST (HUTAN)

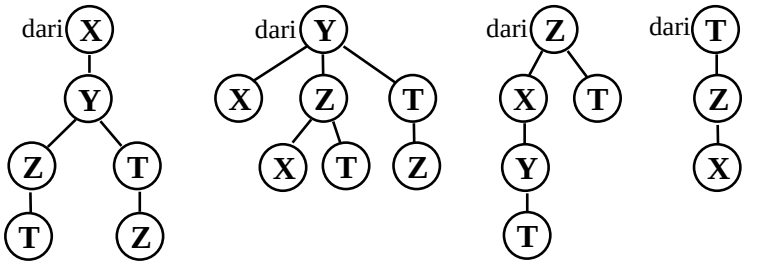
Himpunan beberapa buah tree disebut sebagai FOREST (atau HUTAN).
Forest sering juga disebut sebagai "*Unrooted Tree*" (pohon yang tidak mempunyai akar), karena seakan akan sebuah tree yang akarnya "*hilang*".
Seperti juga tree, forest dapat ditransformasikan menjadi binary tree, dengan "*tambahan aturan* " :

ROOT setiap tree yang menjadi anggota sebuah *FOREST* dianggap sebagai *BROTHERS*.

CONTOH :
misalkan diketahui jaringan jalan raya antara 4 buah lokasi X, Y, Z dan T seperti pada gambar dibawah ini. Kemudian dikehendaki untuk membuat *Himpunan perjalanan* dari setiap lokasi ke lokasi yang lainnya, dengan syarat : tidak ada lokasi yang dilewati *lebih dari 1x.*

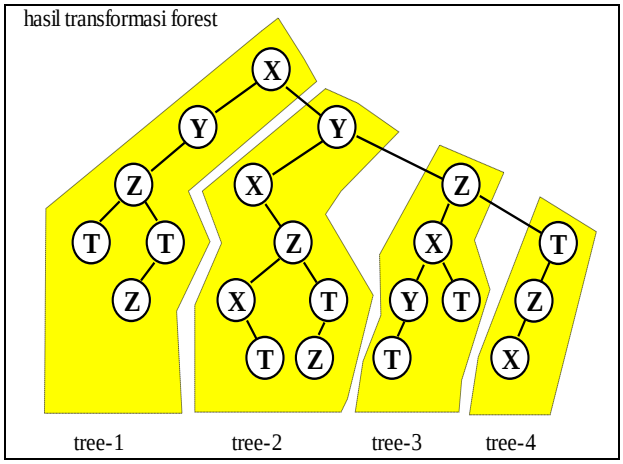


himpunan perjalanan (setiap node 1X dikunjungi)



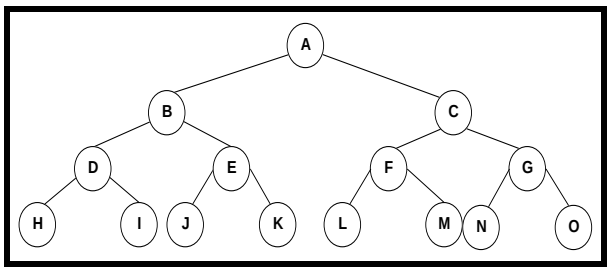
tree-1 tree-2 tree-3 tree-4

Himpunan perjalanan dari A, B, C dan D masing masing menghasilkan sebuah tree seperti pada gambar. Maka dengan demikian seluruh himpunan perjalanan yang dicari tersebut berupa himpunan 4 buah tree tersebut diatas, atau sebuah *forest dengan 4 buah tree*. Jika forest ini ditransformasikan menjadi binary tree, maka diperoleh bentuk seperti pada gambar disamping ini.



4.4. REPRESENTASI dengan ALOKASI STATIS.

Dikarenakan jumlah anak yang bervariasi dalam bentuk tree yang non binary, maka dalam pembahasan kali ini alokasi statis dipakai untuk struktur *Binary Tree* saja.



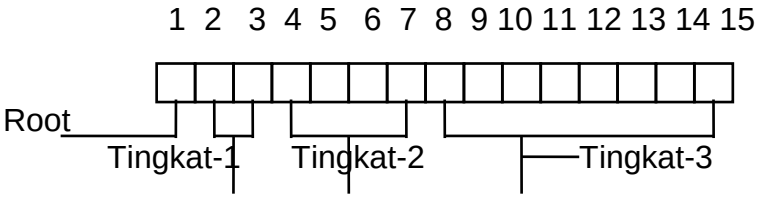
Dari struktur binary tree di atas, maka diperlukan array dengan 15 elemen untuk menyimpannya, dengan aturan :

- Root selalu berada pada elemen pertama;
- Simpul-simpul pada tingkat ke-1 berada di elemen ke-2 sampai dengan elemen ke-3;
- Simpul-simpul pada tingkat ke-2 berada di elemen ke-4 sampai dengan elemen ke-7;
- Simpul-simpul pada tingkat ke-3 berada di elemen ke-8 sampai dengan elemen ke-15.

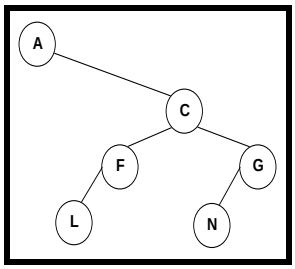
Secara umum :

1. Simpul pada tingkat ke - i terletak pada array dengan index 2^{i} sampai dengan 2^{I+1} -1;
2. Anak-anak elemen ke - i adalah (2*i) dan (2*i + 1) dan parent dari elemen ke - i adalah (i div 2) atau $\lfloor i/2 \rfloor$

Ilustrasi penyimpanan dalam Array :



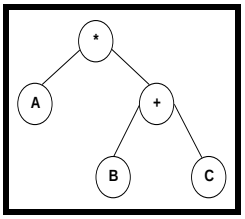
Permasalahan : Penyimpanan jadi tidak hemat jika *binary tree*nya seperti berikut :



karena masih diperlukan array sebanyak (2^{4} - 1) = 15 elemen.

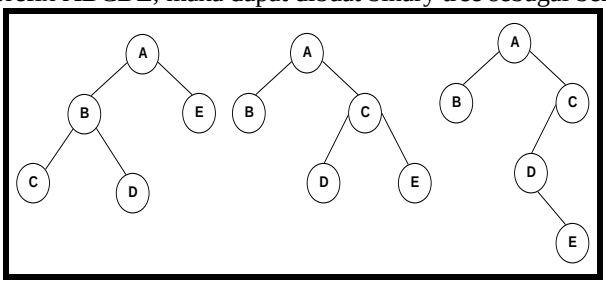
Secara umum *binary tree*, dengan berbagai jenis traversal dapat disajikan secara sequensial,misalnya :

Kasus 1:



Notasi Infix : A*B+C
Notasi prefix : *A+BC
Notasi postfix : ABC+*

Kasus 2 :
Jika diberikan notasi prefix ABCDE, maka dapat dibuat binary tree sebagai berikut :



Dalam traversal preorder, untuk ekspresi matematika (=Kasus 1), OPERATOR menjadi ROOT dan OPERAND menjadi CABANG.
Dalam Kasus 2 tidak diketahui mana yang menjadi ROOT dan mana yang menjadi CABANG, sehingga menghasil-kan banyak variasi tree yang terbentuk. Untuk itu, diperlukan informasi lain yaitu jumlah anak (derajat) dari setiap simpul.
Contoh traversal preorder dan jumlah anaknya :

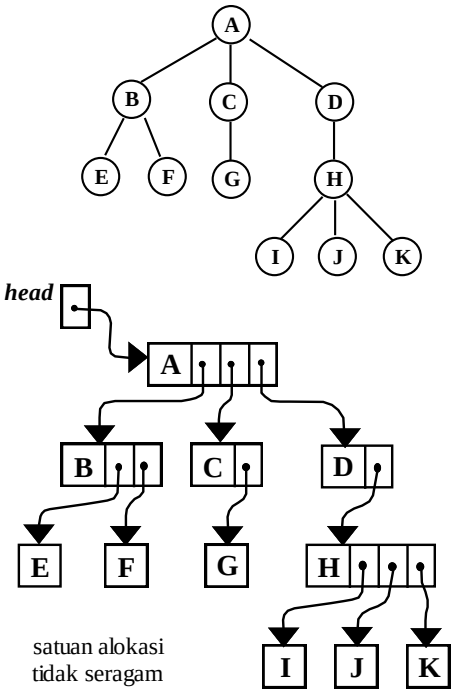
Preorder :	A	B	C	D	E	F	G	H	I	J	K	L
Jml. Anak:	4	0	3	0	0	0	0	2	2	0	0	0

Preorder :	P	Q	R	S	T	U	V	W	X	Y	Z
Jml. Anak:	3	2	0	0	4	0	1	0	0	0	0

Representasi dalam arraynya adalah dengan mengambil simpul yang memiliki anak dan terkanan, secara rekursif.

4.5. REPRESENTASI dengan ALOKASI DINAMIS.

REPRESENTASI TREE dengan menggunakan LINKED LIST



Perhatikan tree disamping ini.
Misalnya ingin dilakukan representasi dengan cara alokasi dimana untuk setiap node pada tree selain data disediakan pointer yang menunjuk anak ke 1, ke 2 dst; disediakan *tepat sejumlah anak* yang dimilikinya.
Maka bentuk representasinya dapat dilihat pada gambar.
Perhatikan adanya pointer *head* yang menunjuk lokasi node A (root), pointer ini berfungsi sebagai "*titik masuk*" pada struktur tree ini.
Pada strategi alokasi seperti ini, *alokasi per node tidak seragam*, tergantung dari jumlah anak. Akan terjadi kesulitan sbb :

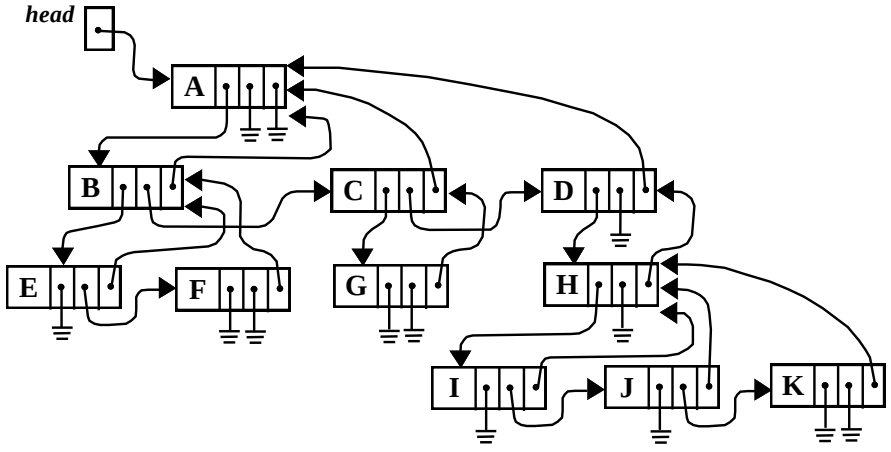
- pada "insertion" (penambahan) atau "deletion" (pemecatan), dimana jumlah anak mungkin berubah.
- pada proses proses lain (mis. traversing) terjadi kesulitan dalam pembuatan algoritmanya.

Oleh karena itu strategi semacam ini tidak lazim dipergunakan untuk merepresentasikan sebuah tree.

Cara yang lebih lazim ditempuh adalah dengan *alokasi yang seragam* per node seperti misalnya pada gambar disamping ini. Dalam hal ini dinamika keanggautaan tree dapat dilayani tanpa harus melakukan alokasi baru atau dealokasi. Akan tetapi terlihat pula bahwa banyak pointer yang bernilai *NIL*, dan ini dianggap suatu bentuk pemborosan.

satuan alokasi per node seragam

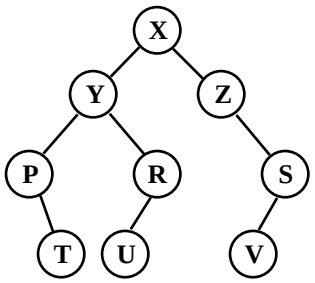
data/info pada node ybs	pointer FIRST-SON	pointer NEXT-BROTHER	pointer FATHER

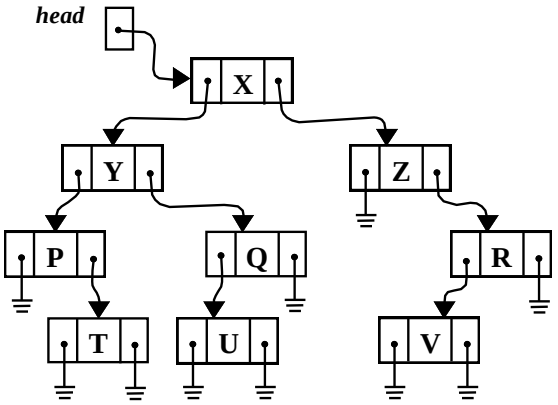


Dapat pula dilakukan alokasi dengan hanya 2 buah pointer, ialah *First-Son* dan *Next-Brother* saja, sedangkan Father tidak. Atau bahkan dipilih cara representasi dimana satuan alokasi per nodenya mempunyai 5 buah pointer : *First-Son, Last-Son, Next-Brother, Prior-Brother dan Father*. Makin banyak pointer berarti makin banyak space yang dibutuhkan, akan tetapi biasanya kecepatan dan kemudahan proses juga bertambah. Pemilihan bentuk representasi merupakan masalah optimasi antara *space* dan *speed* yang sangat tergantung dari *situasi aplikasi* yang dihadapi.

REPRESENTASI BINARY TREE dengan mempergunakan LINKED LIST

Salah satu bentuk representasi yang lazim dipergunakan ialah dengan cara alokasi dimana setiap node mempunyai 2 pointer : *Left-Son* dan *Right-Son*; disamping data/informasi.
Lihat gambar contoh bentuk binary tree dan representasinya.
Perhatikan bahwa banyak pointer yang bernilai NIL ! (berapa prosen dari keseluruhan ?) Perhitungkan pula bahwa makin tinggi atau dalam binary tree tersebut, maka makin banyak leaves, berarti makin besar prosentase pointer yang bernilai nil.





satuan alokasi

pointer LEFT-SON	data/info pada node ybs	pointer RIGHT-SON

Untuk memanfaatkan pointer yang bernilai NIL tersebut, maka dilakukan cara yang disebut "THREADED BINARY TREE" (pohon biner yang dianyam).

Representasi Binary tree dengan Cara Threaded

Satuan alokasi per node untuk *Threaded-Binary-Tree* hampir sama dengan pada cara representasi linked biasa, hanya ada penambahan 2 buah variabel logika ialah : *Left-Thread* dan *Right-Thread* yang hanya mungkin bernilai False atau True.
Cara pemberian nilai setiap variabel adalah sbb :

satuan alokasi cara threaded

LEFT-THREAD	data/info pada node ybs	RIGHT-THREAD
pointer LEFT-SON		pointer RIGHT-SON

1. Untuk setiap node yang mempunyai Left/ Right-Son nilai *pointer Left/ Right-Son sama seperti pada cara linked biasa*; sesuai dengan itu maka *variabel Left/ Right Thread* pun diberi nilai *False*.

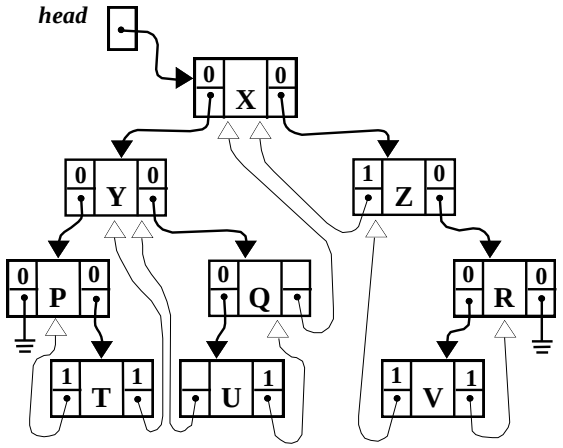
2. Untuk node yang tidak mempunyai Right-Son:

   - pointer Right-Son menunjuk ke node berikut pada urutan traversing Inorder ("*inorder successor*").
   - isi Right-Thread dengan nilai *True*.

3. Untuk node yang tidak mempunyai Left-Son :

   - pointer Left-Son menunjuk ke node tepat sebelumnya pada urutan traversing Inorder ("*inorder predecessor*").
   - isi Left-Thread dengan nilai *True*.

Maka untuk binary tree seperti pada contoh diatas, bentuk threaded nya dapat dilihat pada gambar disamping ini. Perhatikan bahwa pointer yang semula bernilai NIL sekarang termanfaatkan, kecuali untuk node *P* (pointer *Left-Son*, karena *tidak ada node sebelum P pada urutan inorder*) dan *R* (pointer *Right-Son*, karena *tidak ada node sesudah R pada urutan inorder*).



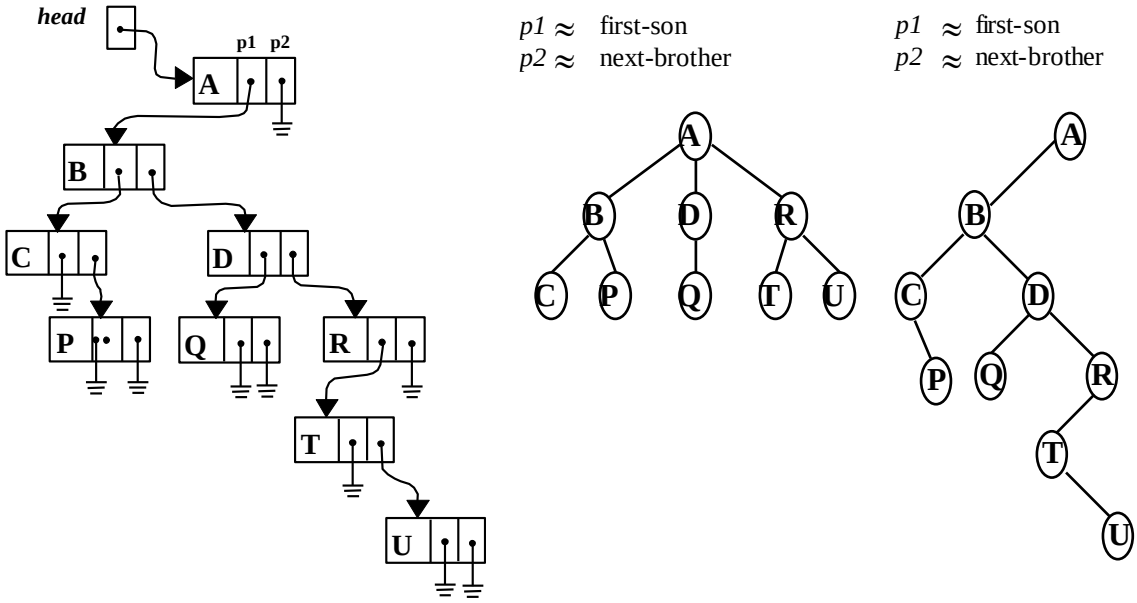
catatan : 0 berarti False, 1 berarti True.

Bahasan Transformasi & Representasi

Perhatikan contoh suatu representasi struktur tertentu seperti pada gambar dibawah ini. Setiap node pada gambar mempunyai sebuah field data dan 2 buah field pointer, sebut saja *p1* dan *p2*.
Dilakukan 2 macam penafsiran terhadap arti dari p1 dan p2 sbb :

a) p1 ≈ pointer *first-son*, dan p2 ≈ pointer *next-brother*. Ini sama saja dengan menganggap bahwa struktur detail pada gambar tersebut adalah representasi dari struktur *tree* biasa.

b) p1 ≈ pointer *left-son*, dan p2 ≈ pointer *right-son*. Dalam hal ini struktur detail dianggap representasi dari struktur *binary-tree*.



Kemudian jika kita amati kedua gambar tree yang terjadi, ternyata bahwa jika tree hasil butir (a) kita transformasikan menjadi binary-tree korespondensnya, maka akan diperoleh gambar binary-tree hasil butir (b). Maka dapat disimpulkan bahwa meskipun secara konseptual dilakukan transformasi tree menjadi binary tree, secara representatif *bentuk detail tidak berubah*. Yang berubah hanya *penafsiran terhadap fungsi pointer* p1 dan p2 !