Binary Tree

(TIB11 – Struktur Data)

Pertemuan 19, 20

Sub-CPMK

- Mahasiswa mampu membentuk binary tree dan melakukan penelusuran binary tree (C3, A3)

Materi:

1. Pengertian *Binary Tree*
2. Menambah Simpul
3. *Binary Tree Traversal*
4. Menemukan Induk Node

1. Pengertian *Binary Tree*

1.1. Trees

- *Tree* adalah Struktur data yang diakses mulai dari simpul akar sampai ujung-ujung daun.
- *Tree* merupakan *graph* terhubung yang berurutan, tidak berputar dan tidak berarah.

1.1. Trees (Lanj.)

- Setiap simpul berupa daun atau simpul internal.
- Simpul internal memiliki satu atau lebih simpul anak dan disebut induk dari simpul anaknya.
- Semua anak dari simpul yang sama adalah saudara.
- *Tree* pada struktur data berlawanan dengan pohon secara fisik, akarnya biasanya digambarkan di bagian atas struktur, dan daunnya digambarkan di bagian bawah.

1.1. Trees (Lanj.)

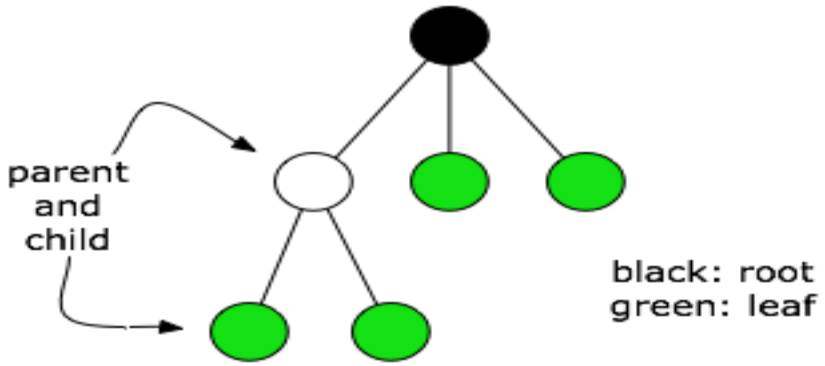


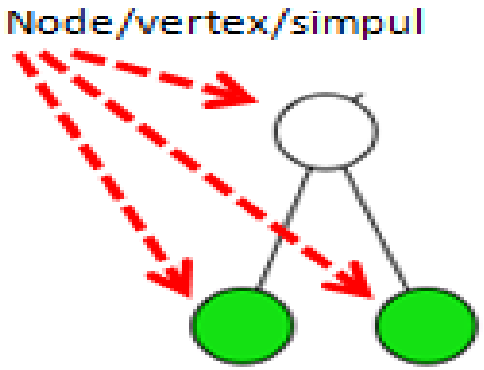
Figure: tree data structure

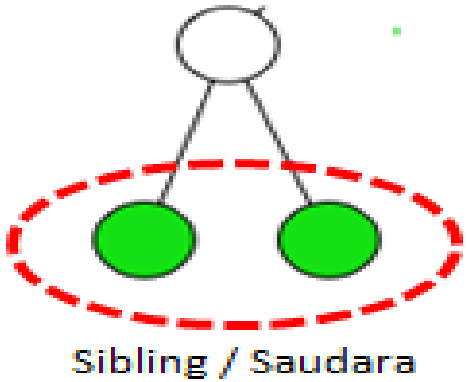
- Terdiri dari nodes/simpul dan panah
- Digambarkan terbalik dengan akar di bagian atas dan daun di bagian bawah

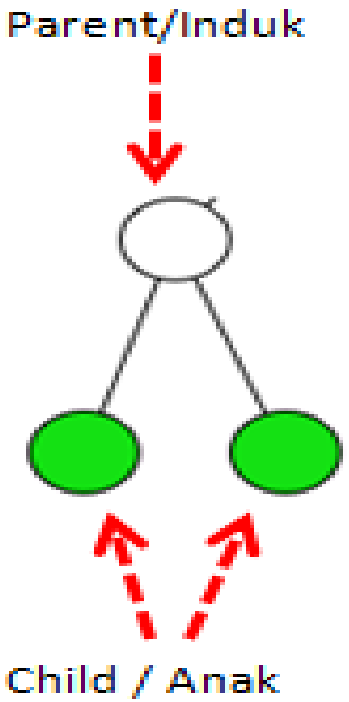
1.2. Terminologi

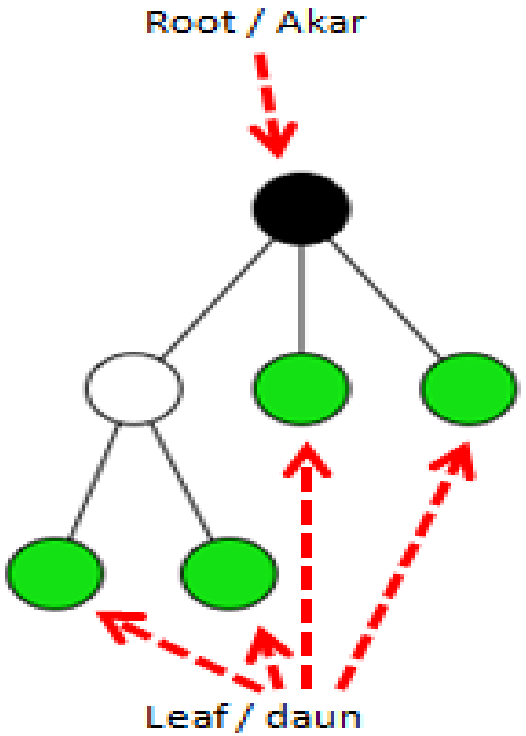
- Node / Vertex / Simpul
  - Referensi pada struktur data
  - Sekumpulan informasi yang berada pada sebuah lokasi memory
- *Parent*/Induk
- *Child*/Anak
- *Sibling*/Saudara
- *Descendant* / keturunan
- *Ancestor* / leluhur
- *Root* / Akar: node/simpul paling awal, hanya satu *item* dan tidak punya induk
- *Leaf* / Daun / Internal node: Node/simpul pada *tree* yang paling ujung / tidak punya anak

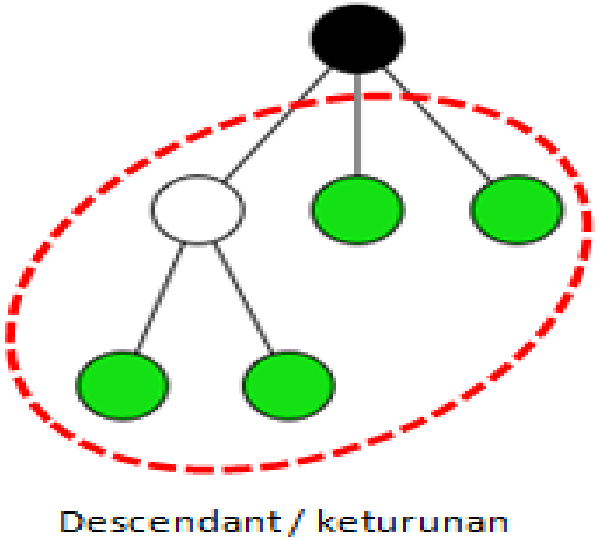
1.2. Terminologi (Lanj.)

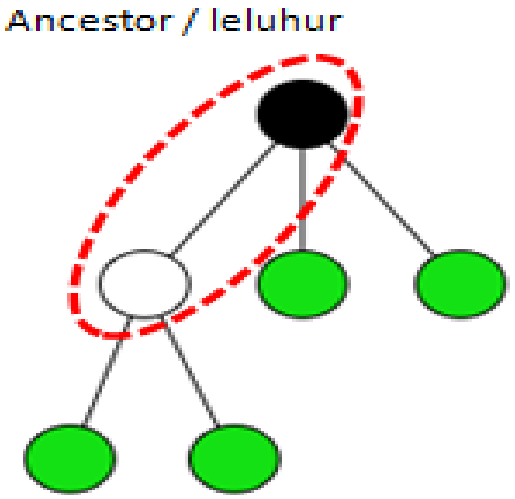








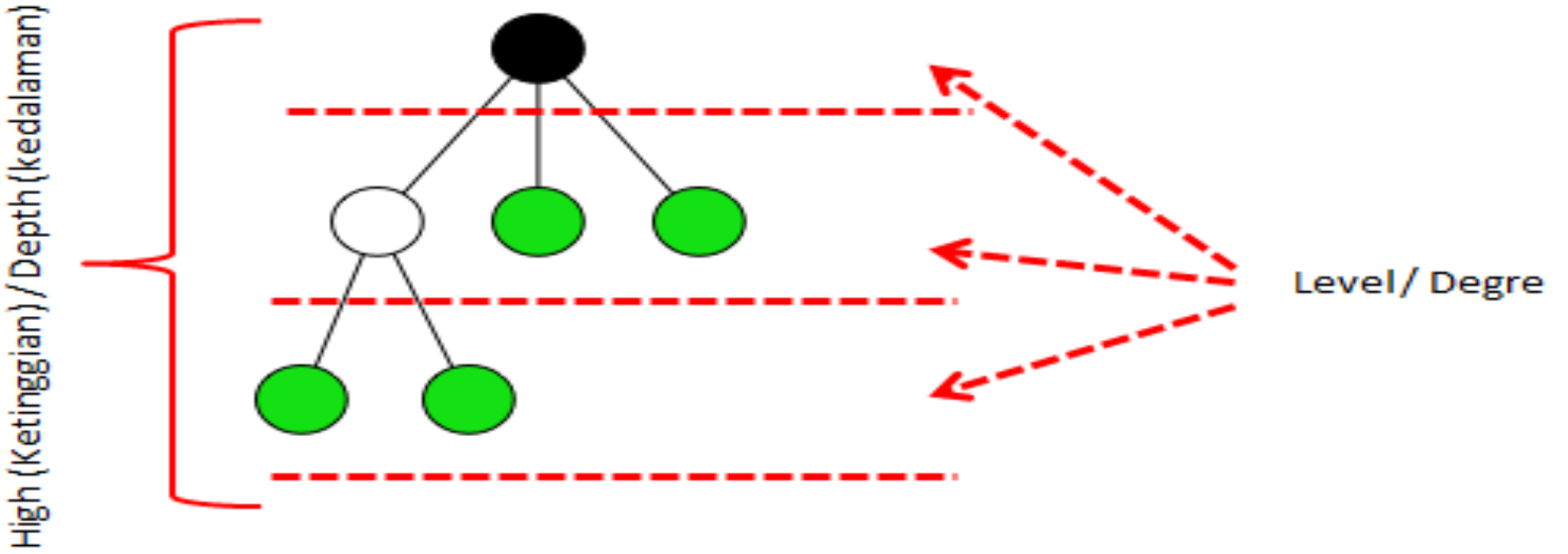


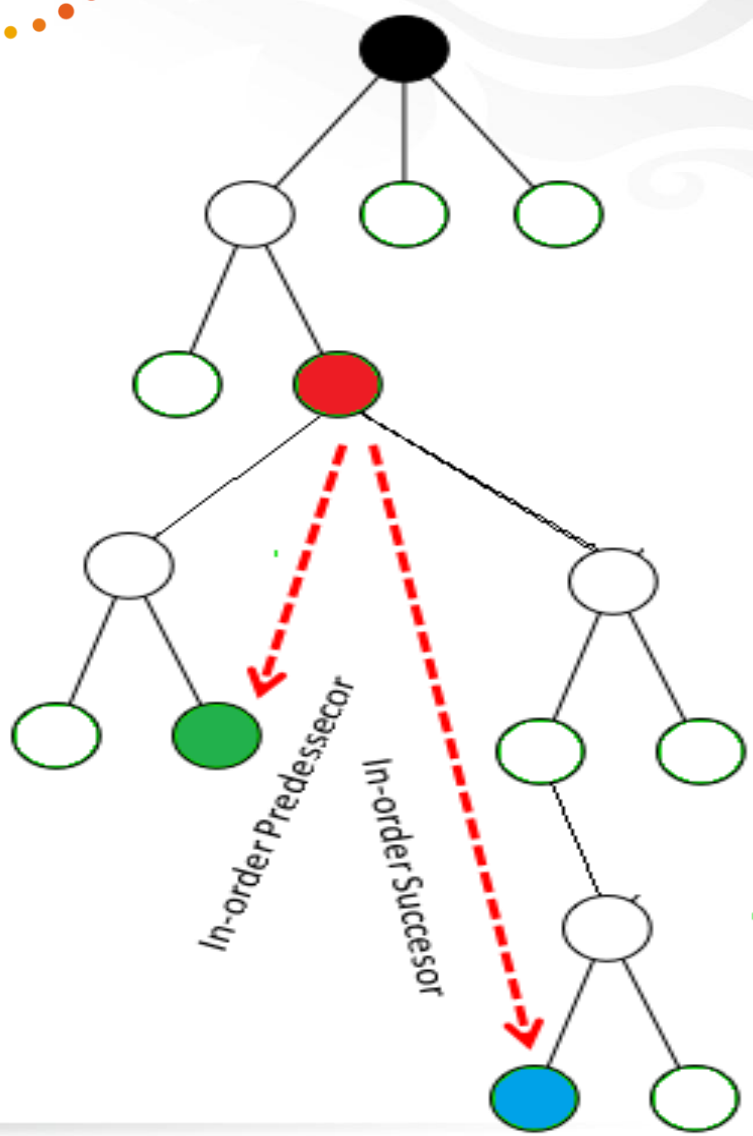


1.2. Terminologi (Lanj.)

- Level / *Degree*
- *Height* (ketinggian) / *depth* (kedalaman)
- Predesesor (*intermediate predecessor*) : Sel pendahulu, dalam *tree traversal* adalah yang dikunjungi terlebih dahulu
- Suksesor (*Successor*): sel berikut, dalam *tree traversal* adalah sel berikut yang akan dikunjungi

1.2. Terminologi (Lanj.)





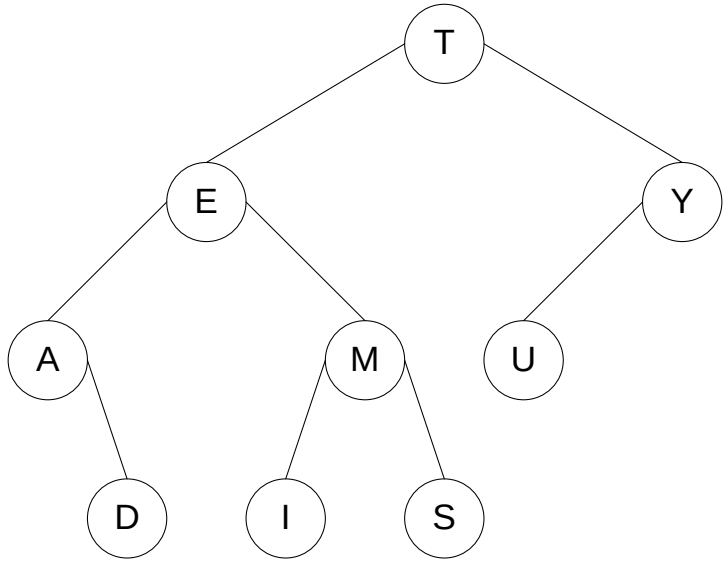
1.3. Jenis *Tree*

Dua jenis *tree*

- *Binary Tree*
- *Multiary Tree* (n-Ary *Tree)*

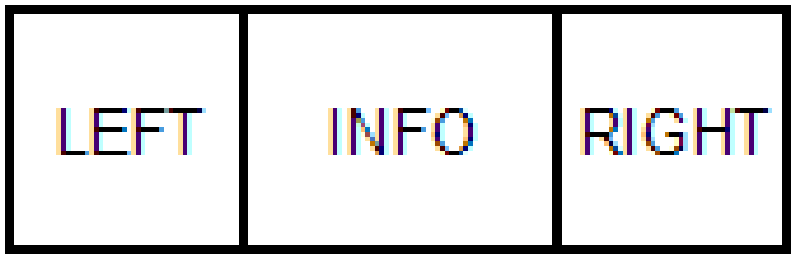
1.4. *Binary Tree*

- *Tree* dengan paling banyak dua anak pada tiap node nya disebut *binary Tree*.

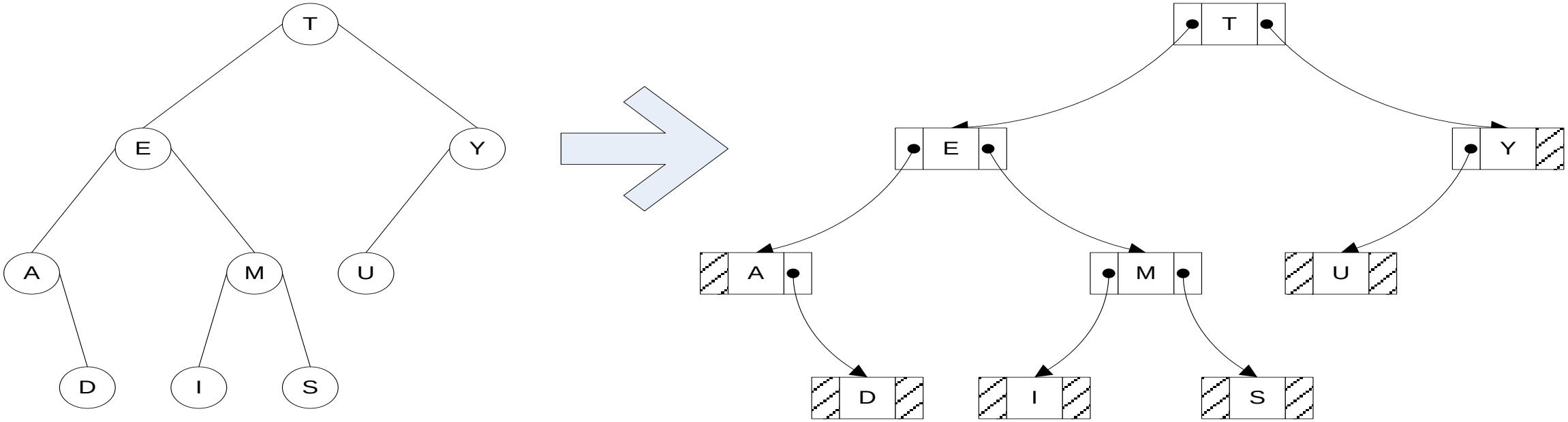


1.5. Implementasi *Binary Trees* Dengan *Linked List*

- *Element*
  - *Information*
  - *Left link → link to the child node at the left side*
  - *Right link → link to the child node at the right side*



1.5. Implementasi *Binary Trees* Dengan *Linked List (Lanj.)*



2. Menambah Simpul

- Menempatkan sebuah *key* baru pada *binary tree*
- Menambah simpul ada dua orientasi yaitu *left-to-Right* dan *Right-to-Left*
- Untuk *Left-to-Right*: Setiap *key* pada kanan *subtree* harus lebih besar dari setiap *key* di kiri *subtree* (berlaku kebalikannya untuk *Right-to-Left*)
- *Rules*:
  - Jika *key* baru lebih besar dari node induk, arahkan ke node anak sebelah kanan
  - Jika *key* baru lebih kecil dari node induk, arahkan ke node anak sebelah kiri
  - Ulangi proses di atas sampai didapatkan node yang tidak memiliki *leaf*
  - Tempatkan *key* baru pada sebagai kanan atau kiri node berdasarkan dua kondisi pertama di atas

3. *Binary Tree Traversal*

3.1. *Tree Traversal*

- Proses mengunjungi setiap simpul pada *tree*/pohon biner tepat satu kali untuk setiap node
- Dapat juga ditafsirkan sebagai meletakkan semua node pada satu baris atau linearisasi *tree*/pohon
- Dapat dilakukan dari kiri ke kanan ataupun dari kanan ke kiri.
- Ada tiga metode *Traversal*:
  - *Pre-Order Traversal*
  - *In-Order Traversal*
  - *Post-Order Traversal*

3.2. Notasi *Tree Traversal*

- V – *Visiting a node* (mengunjungi Simpul)
- L – Melintasi *subtree* ke kiri (*Left*)
- R – Melintasi *subtree* ke kanan (*Right*)

3.3. *Left To Right Traversal*

Mengunjungi setiap *node* pada *tree* secara rekursif pada kiri dan kanan *subtrees* suatu simpul.

- *Preorder*: V-L-R
  - *visiting the node*
  - *traversing the left subtree*
  - *traversing the right subtree*
- *Postorder*: L-R-V
  - *traversing the left subtree*
  - *traversing the right subtree*
  - *visiting the node*.
- *inorder*: L-V-R
  - *visiting the left subtree*
  - *visiting the node*
  - *traversing the right subtree*.

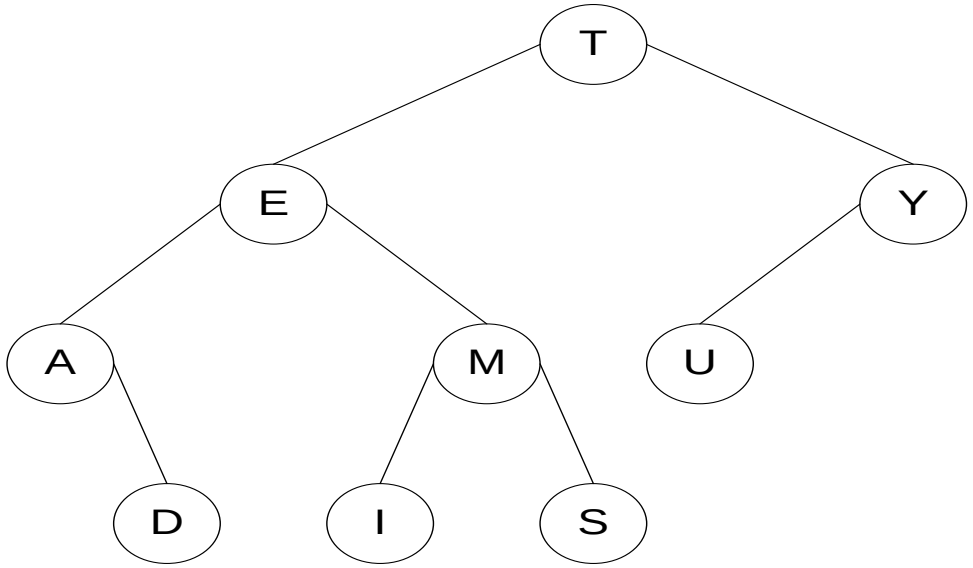
3.4. *Right to Left Traversal*

Kebalikan dari *Left-to-Right*, proses *traversal* dilakukan secara rekursif dari kanan ke kiri

- *Preorder*: V-R-L
  - *visiting the node*
  - *traversing the right subtree*
  - *traversing the left subtree*
- *Postorder*: R-L-V
  - *traversing the right subtree*
  - *traversing the left subtree*
  - *visiting the node.*
- *inorder*: R-V-L
  - *traversing the right subtree.*
  - *visiting the node*
  - *visiting the left subtree*

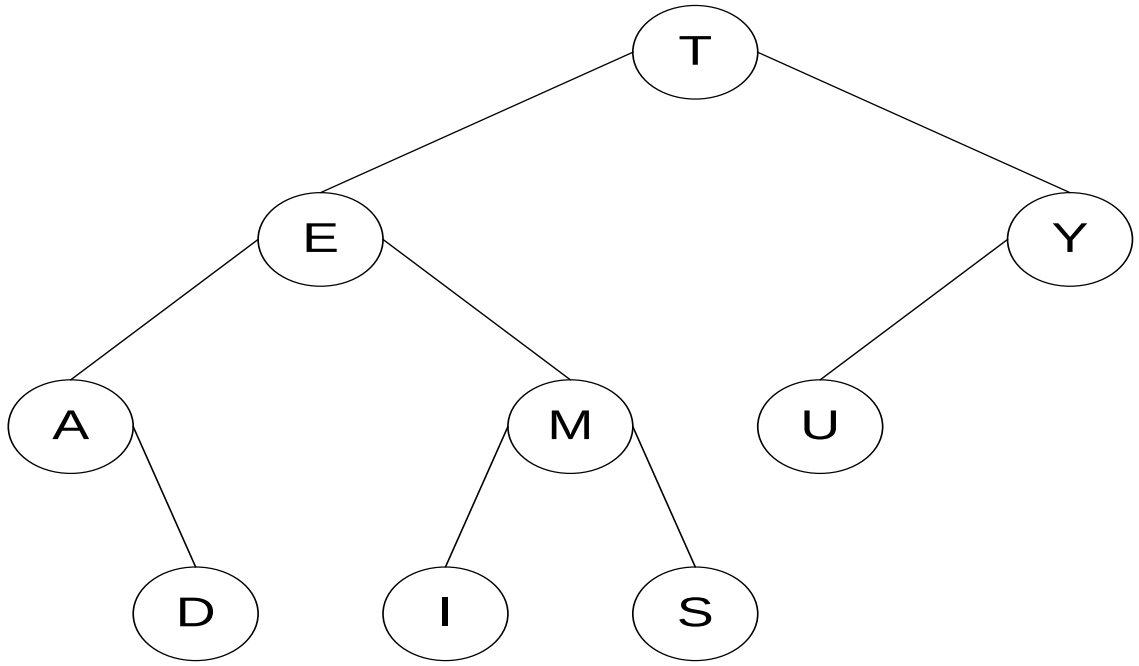
Pembahasan Tree Traversal berikut ini berdasarkan Left-To-Right Traversal, untuk penggunaan Right-To-Left Traversal dapat menyesuaikan

3.5. *Pre-order Traversal*



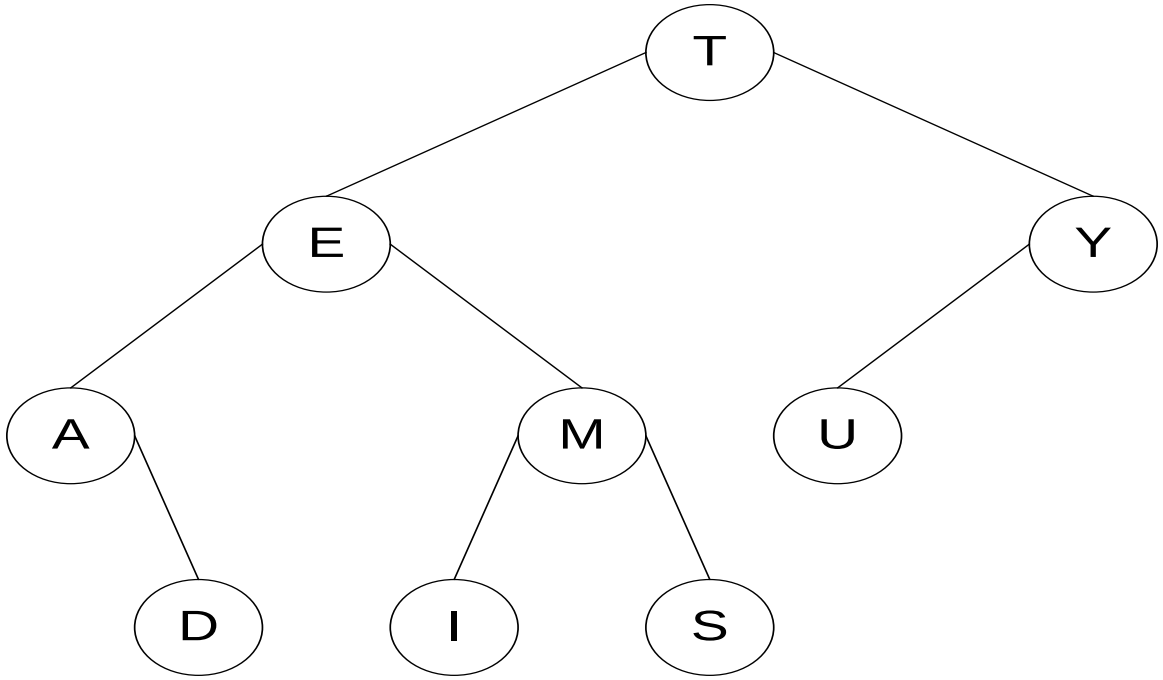
root - Visiting T – go left (E) - Visiting E – go left (A) – visiting A – go right (D) – visiting D – go parent (A) – go parent (E) - go right (M) – visiting M – go left (I) - visiting I – go parent (M) – go right (S) – visiting S – go parent (M) – go parent (E) – go parent (T) – go right (Y) - visiting Y – go left (U) – visiting U – finish

3.6. *Post-order Traversal*



root - go left (E) – go left (A) – go parent A – go right (D) – visiting D – go parent A – visiting A – go parent E – go right (M) – go left (I) – Visiting I - go parent M – go right (S) – visiting S – go parent M – visiting M – go parent E – visiting E – go parent T – go right (Y) – go left (U) – visiting U – go parent (Y) – visiting Y – go parent (T) – visiting T – finish

3.7. *In-Order Traversal*

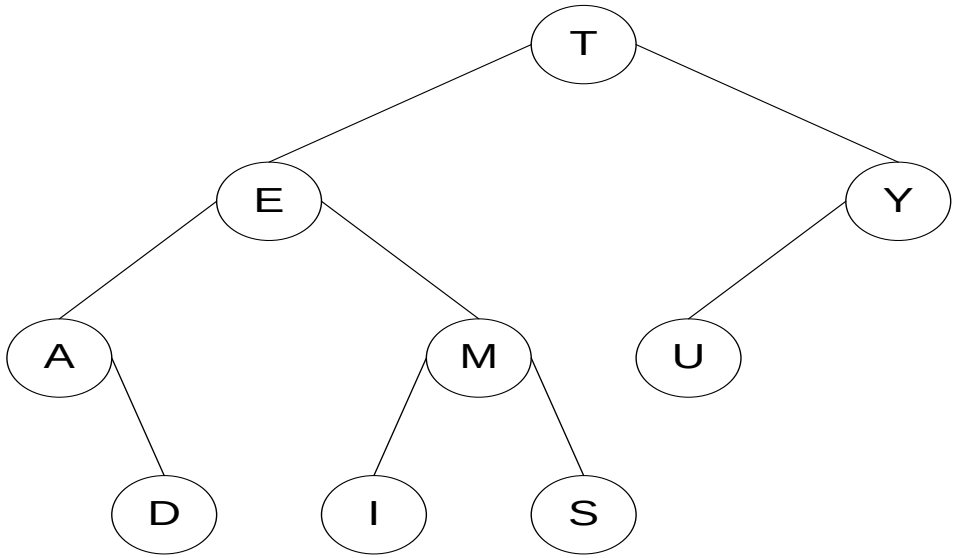


root (T) – go left (E) – go left (A) – visiting A – go right(D) - visiting D – go parent (A) – go parent (E) – visiting E – go right (M) – go left(I) - visiting I – go parent (M) – visiting M - go right (S) – visiting S – go parent (M) – go parent (E) – go parent (T) – visiting T – go right (Y) – go left (U) – visiting U – go parent (Y) – visiting Y – finish

4. Menemukan Induk Node

4.1. *Predecessor* dan *Successor*

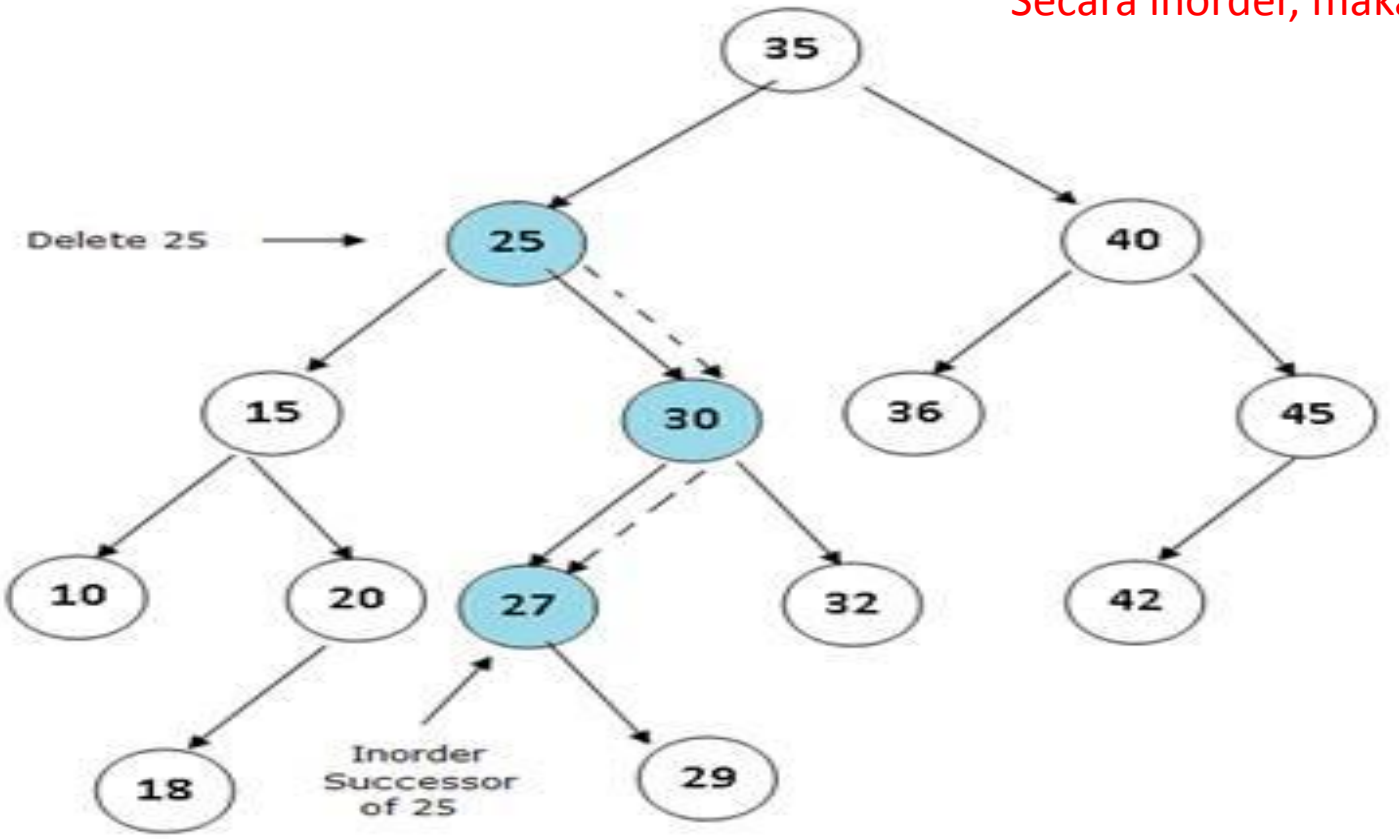
- Berdasarkan dari metode Traversal yang digunakan
- Predesesor (*Predecessor*) adalah node yang baru saja dikunjungi
- Suksesor (*Succesor*) adalah node yang akan dikunjungi
- Contoh pada *Traversal inorder* dari *tree* disamping:
- *Inorder-traversal*: A-D-E-I-M-B-T-U-Y
- Maka predesesor dari T adalah S, sedangkan suksesor dari T adalah U



4.1. *Predecessor* dan *Successor* (Lanj.)

Contoh Inorder Successor of node 25

Secara inorder, maka didapat successor dari 25 adalah 27



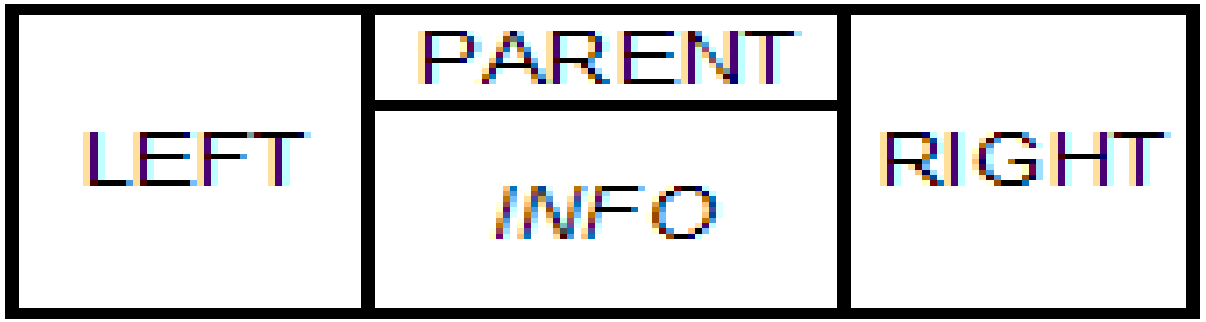
Sumber gambar: http://vle.du.ac.in/mod/book/view.php?id=5726&chapterid=3023

4.2. Menemukan Induk Dari Suatu Node

Dapat dilakukan dengan cara:

- Telusuri dari *root* sampai didapatkan induk dari node yang akan diproses → Pasti berhasil, tapi memakan waktu
- Gunakan *tree* dengan *link* ke induk → memerlukan *memory* yang lebih banyak, tapi setidaknya lebih baik daripada ide di atas
- Gunakan *Threaded Tree*

4.3. *Tree* Dengan Link Induk



- *Binary Tree* dimana setiap node nya memiliki informasi link ke induk
- *Elements*:
  - *Info*
  - *Left Child Link*
  - *Right Child Link*
  - *Parent Link*

4.4. *Threaded Tree*

- Pohon pencarian biner di mana masing-masing *node* menggunakan tautan anak kiri yang kosong untuk merujuk ke *node in-order predecessor* induk dan tautan anak kosong yang kanan untuk merujuk ke *node in-order successor* Induk dari Induk (*grand parent*)

http://www.nist.gov/dads/HTML/threadedtree.html

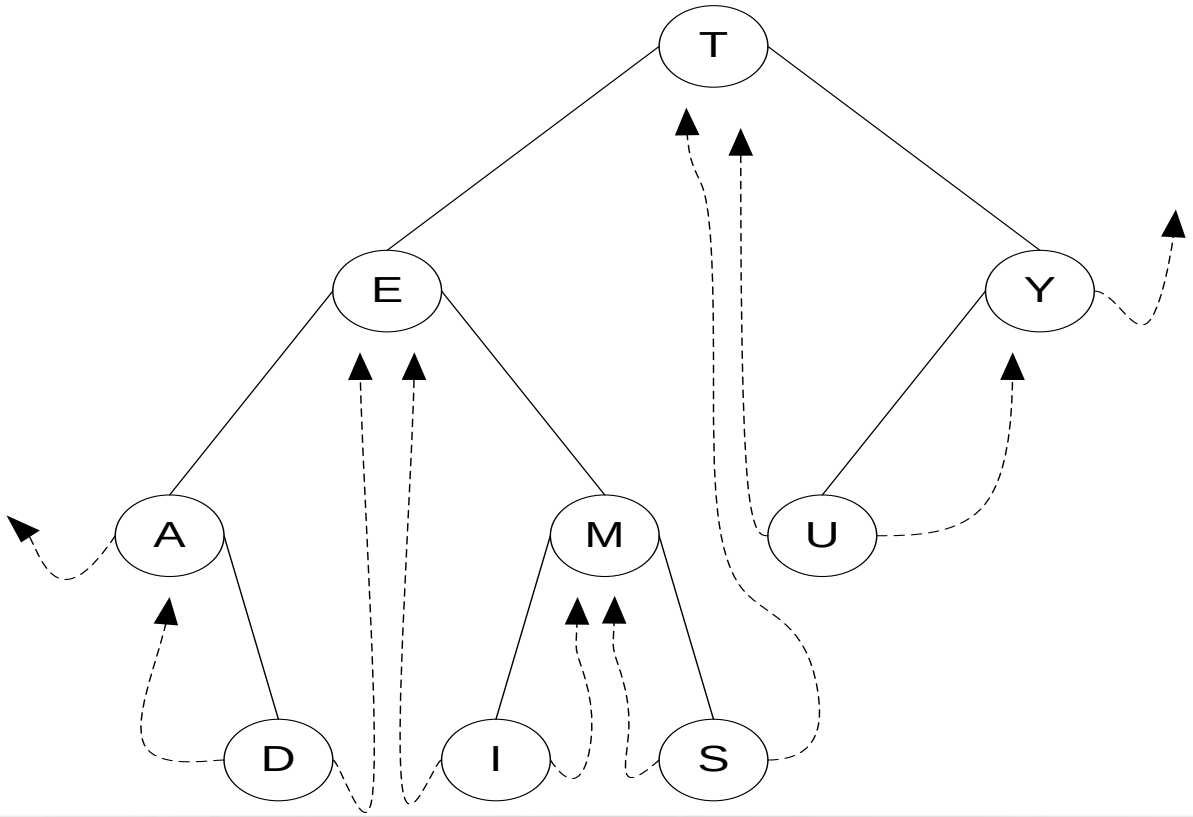
(October 17th 2008)

4.5. *Threaded Tree Elements*

- *Key*
- *Left Link*
- *Threaded Left* → Sebuah nilai boolean untuk mengindikasikan bahwa *left link* adalah *thread* ke *last visited node* (induk *in-order predecessor*)
- *Right Link*
- *Threaded Right* → Sebuah nilai boolean untuk mengindikasikan bahwa *right link* adalah *thread* ke *next node* yang akan dikunjungi (induk dari induk *in-order successor*)

4.5. *Threaded Tree Elements* (Lanj.)

4.6. Contoh *Threaded Tree*



Ringkasan

- *Tree* dengan paling banyak dua anak pada tiap *node* nya disebut *binary Tree*.
- Menambah simpul ada dua orientasi yaitu *left-to-Right* dan *Right-to-Left*
- Untuk *Left-to-Right*: Setiap *key* pada kanan *subtree* harus lebih besar dari setiap *key* di kiri *subtree* (berlaku kebalikannya untuk *Right-to-Left*)
- *Tree Traversal* adalah Proses mengunjungi setiap simpul pada *tree*/pohon biner tepat satu kali untuk setiap node
- Ada tiga metode *Traversal*:
  - *Pre-Order Traversal (VLR)*
  - *In-Order Traversal (LVR)*
  - *Post-Order Traversal (LRV)*

Contoh in order traversal

```
void inOrderTrav(struct TheCell *travCell)
{
  if (travCell->kiri != NULL)
    inOrderTrav(travCell->kiri);
  cout<<travCell->dat<<" | ";
  if (travCell->kanan != NULL)
    inOrderTrav(travCell->kanan);
}
```

Terimakasih

TUHAN Memberkati Anda

Teady Matius Surya Mulyana (tmulyana@bundamulia.ac.id)