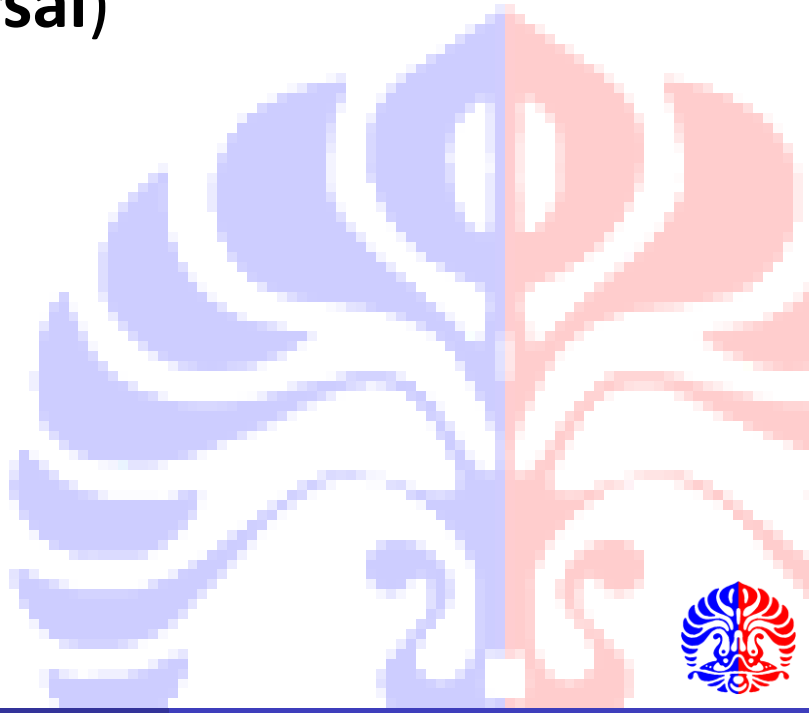
IKI10400 • Struktur Data & Algoritma: Tree

Fakultas Ilmu Komputer • Universitas Indonesia

Slide acknowledgments:
Suryana Setiawan, Ade Azurat, Denny, Ruli Manurung

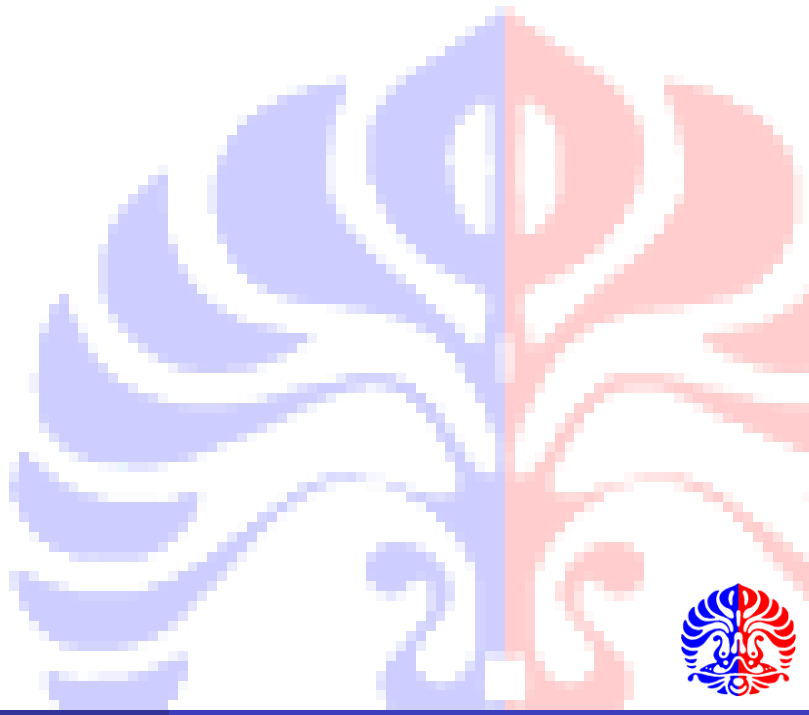
Tujuan

- Memahami definisi dan terminologi mengenai tree secara umum.
- Mengenali aplikasi tree.
- Mengetahui cara melakukan operasi untuk tiap-tiap element pada tree (**tree traversal**)



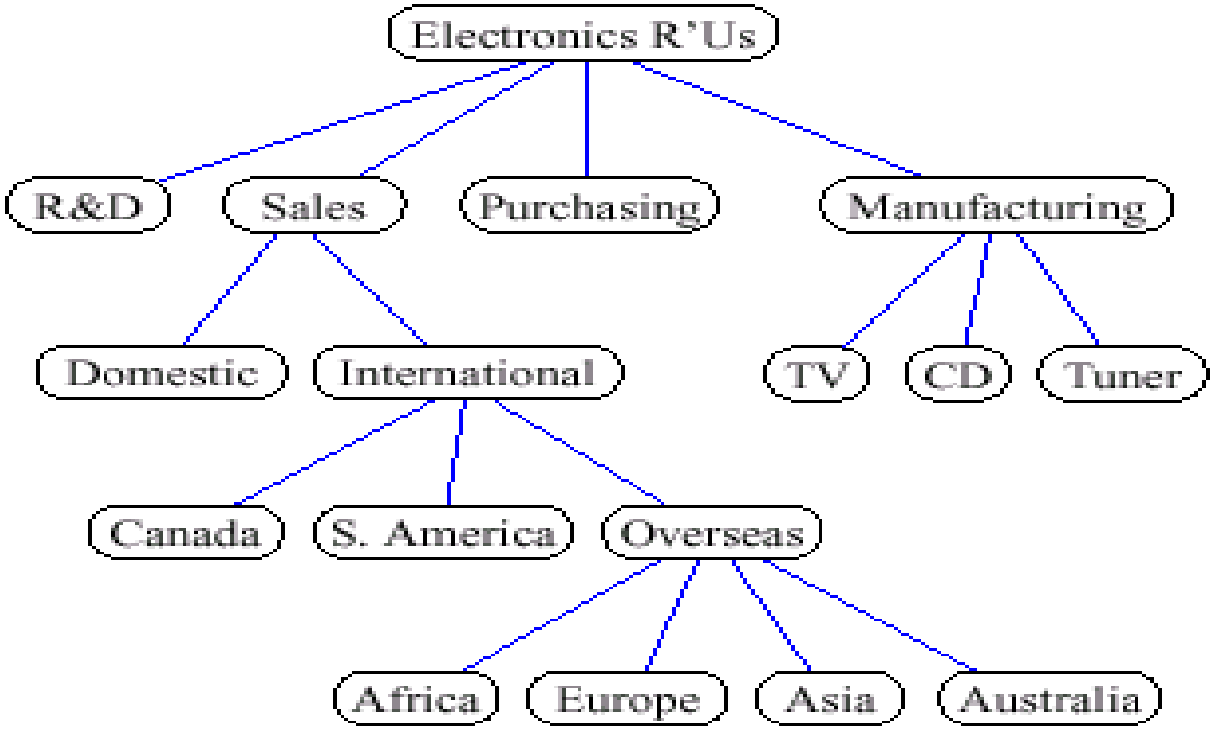
Outline

- Tree
  - Contoh
  - terminologi/definisi
- Binary tree
- Traversal of trees
- Iterator



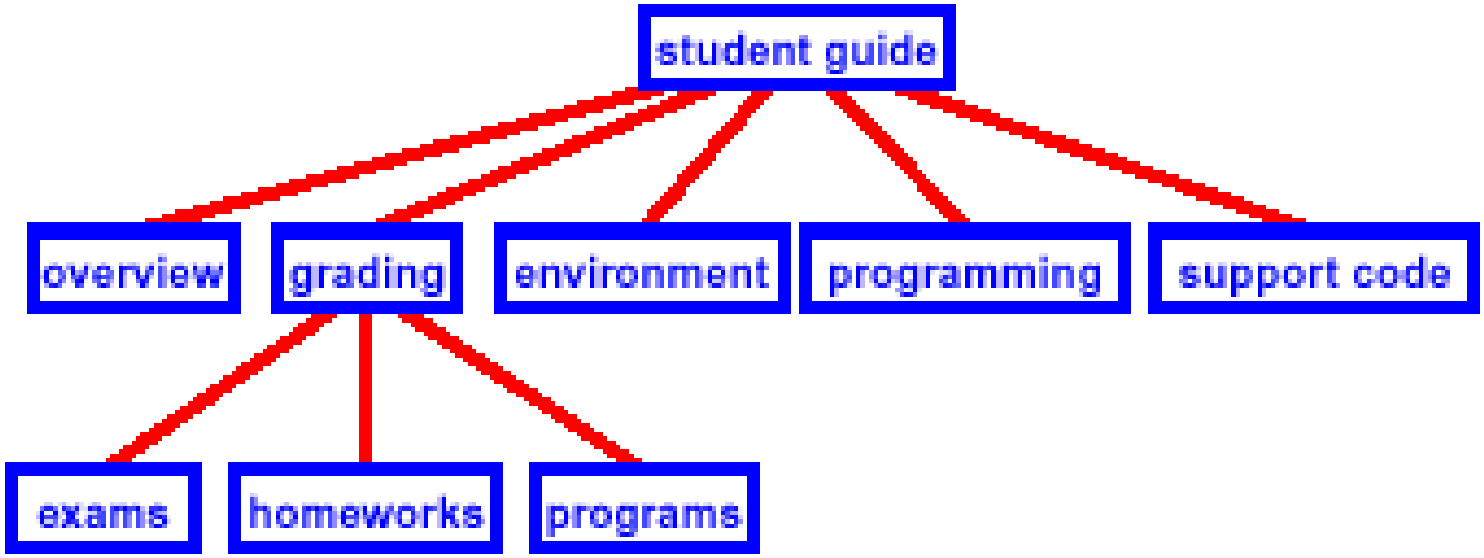
Contoh

- Sebuah tree merepresentasikan sebuah ***hirarki***
  - Mis.: Struktur organisasi sebuah perusahaan



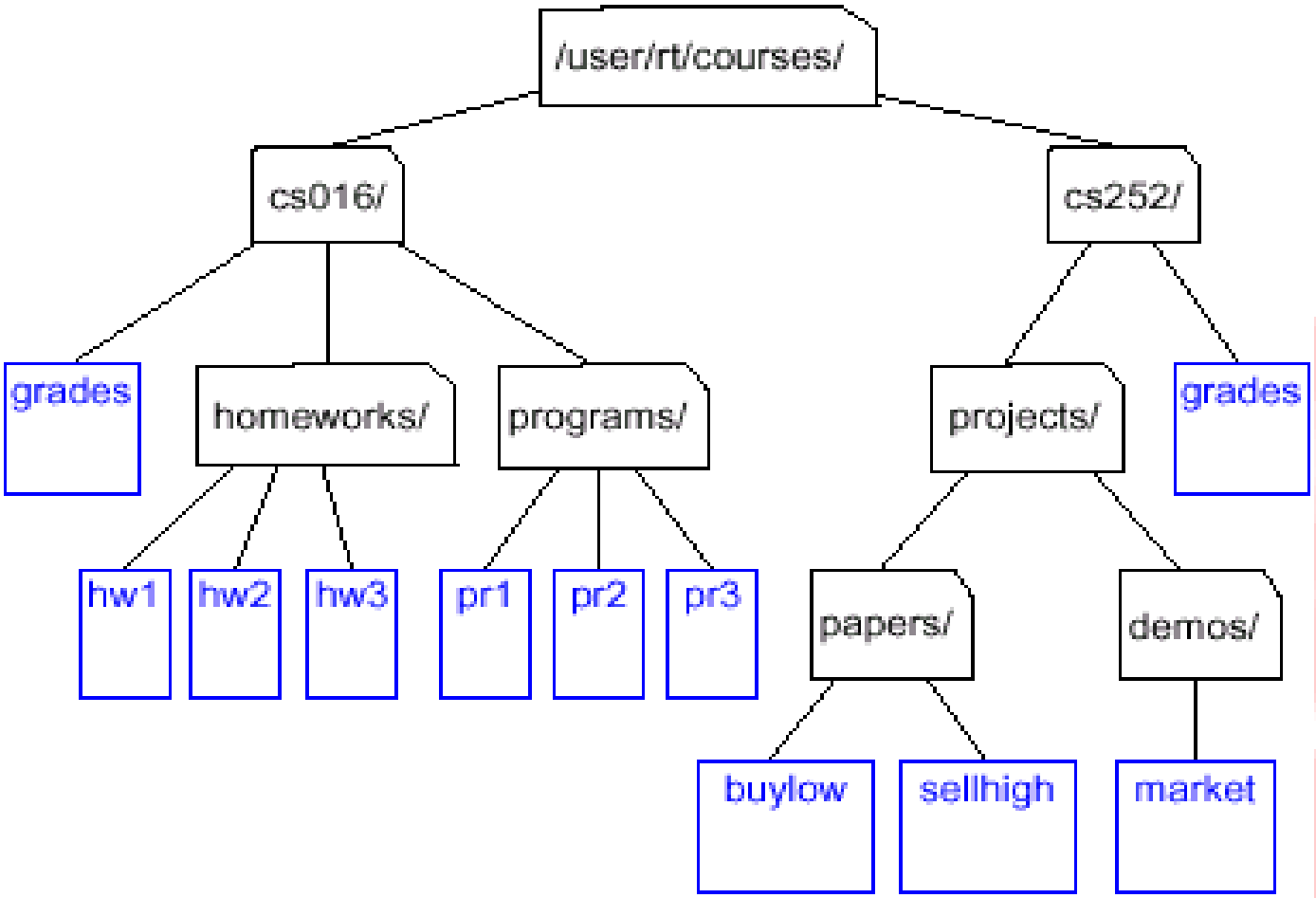
Contoh

- Daftar isi sebuah buku

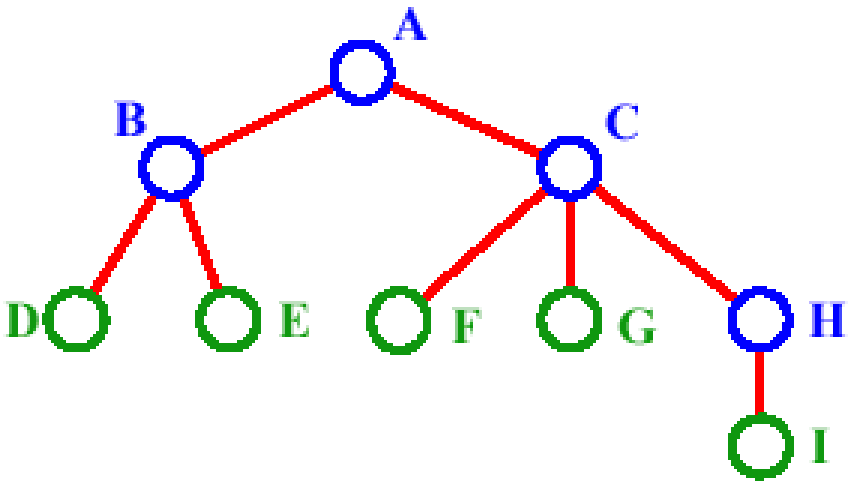


Contoh

- File system

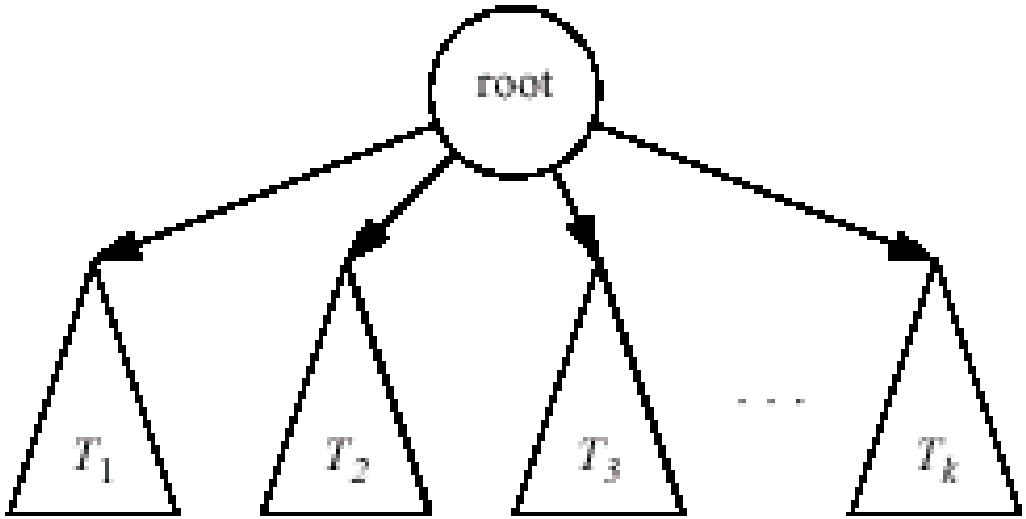


Istilah-istilah umum:



- *A* is the *root node*
- *B* is the *parent* of D and E
- *C* is the *sibling* of B
- *D* and *E* are the *children* of B
- *D, E, F, G, I* are *external nodes*, or *leaves*
- *A, B, C, H* are *internal nodes*
- The *depth/level/path length* of *E* is 2
- The *height* of the tree is 3
- The *degree* of node *B* is 2
- Property:
  (# *edges*) = (#*nodes*) - 1

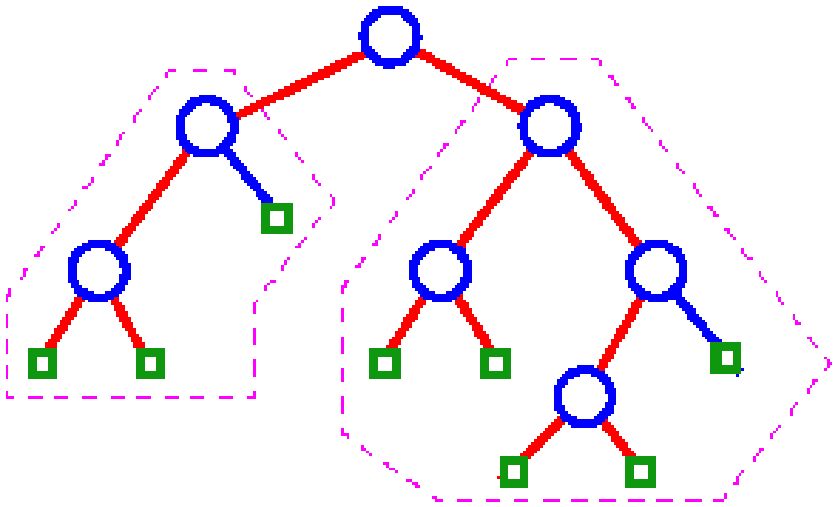
Tree dilihat secara Rekursif



- Setiap *sub-tree* adalah juga sebuah *tree*!

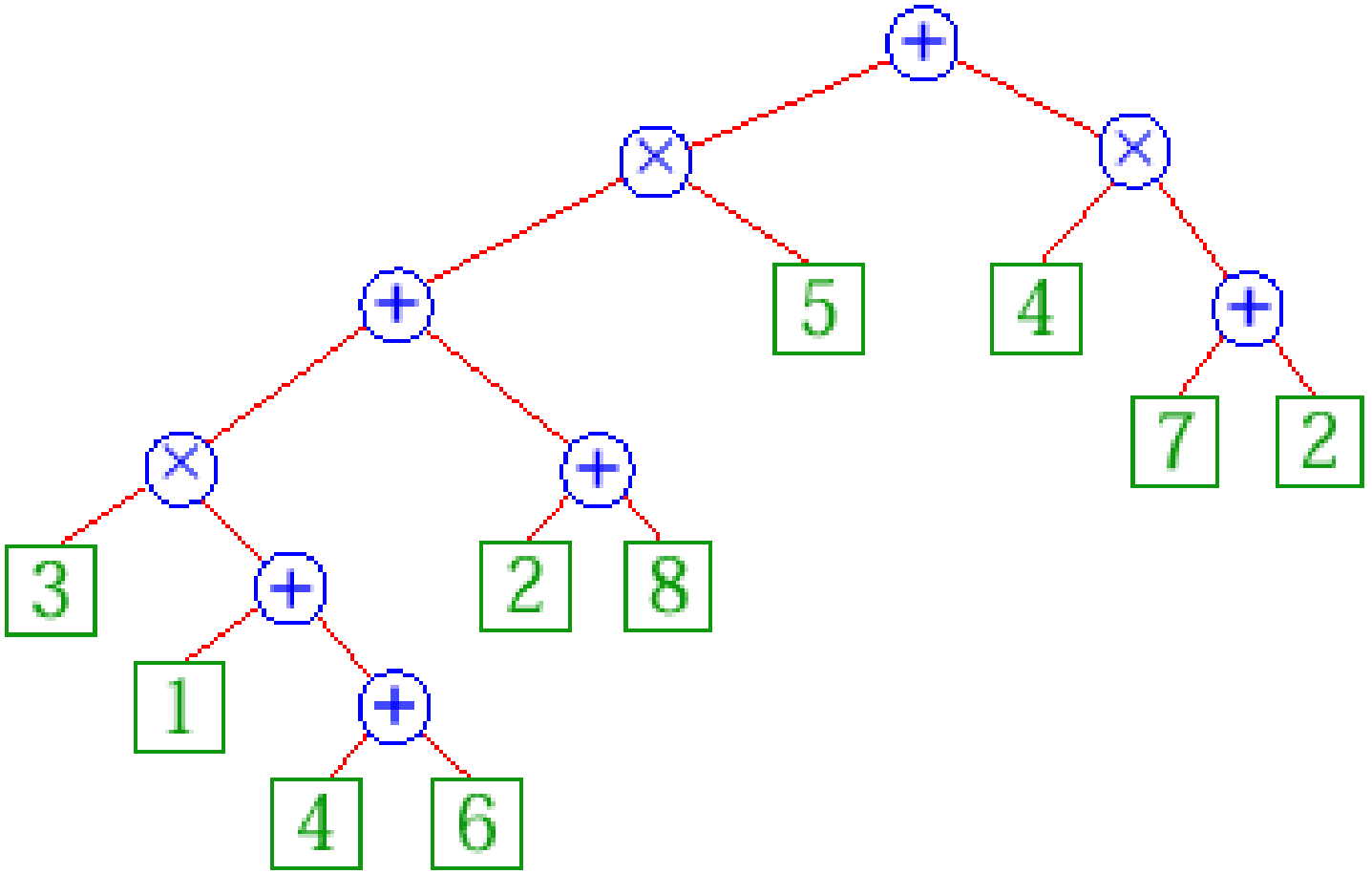
Binary Tree

- *Binary tree:* tree di mana semua ***internal nodes*** memiliki maksimum *degree 2*
- *Ordered/Search tree:* seluruh ***children*** dari tiap ***node*** terurut



Contoh Binary Tree

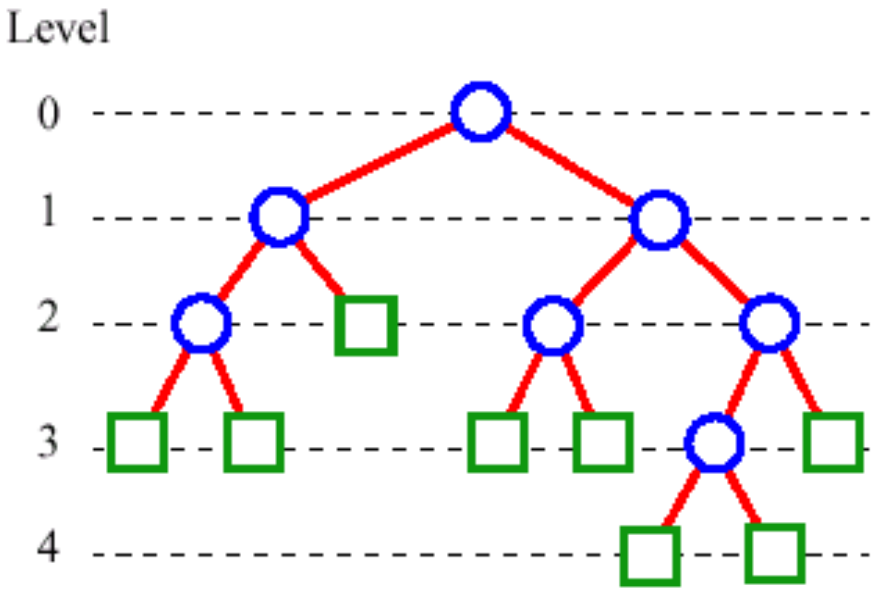
- Representasi ekspresi arithmatik



((((3 × (1 + (4 + 6))) + (2 + 8)) × 5) + (4 × (7 + 2)))

Ketentuan mengenai *Binary Tree*

- Jika dibatas bawah tiap *node* hanya memiliki dua *node* anak (*children*), maka:
  - (# external nodes) = (# internal nodes) + 1
  - (# nodes at level i) $\leq 2^i$
  - (# external nodes) $\leq 2^{(\text{height})}$
  - (height) $\geq \log_2$ (# external nodes)
  - (height) $\geq \log_2$ (# nodes) - 1
  - (height) $\leq$ (# internal nodes) = ((# nodes) - 1)/2



Ketentuan mengenai *Binary Tree*

- Jumlah maksimum node dalam binary tree dengan tinggi *k* adalah $2^{k+1} - 1$.
  - Sebuah **full binary tree** dengan tinggi *k* adalah sebuah binary tree yang memiliki $2^{k+1} - 1$ nodes.
  - Sebuah **complete binary tree** dengan tinggi k adalah binary tree yang miliki jumlah maximum nodes di levels 0 sampai k – 1 (semua level terisi kecuali pada level terakhir, yang terisi dari sisi kiri hingga kanan dan tidak memiliki ***missing nodes***).

BinaryNode dalam Java

- Tree adalah sekumpulan *nodes yang dideklarasikan secara rekursif.*

```
class BinaryNode<A>
{
    A element;
    BinaryNode<A> left;
    BinaryNode<A> right;
}
```

ADT *Tree* dalam Java

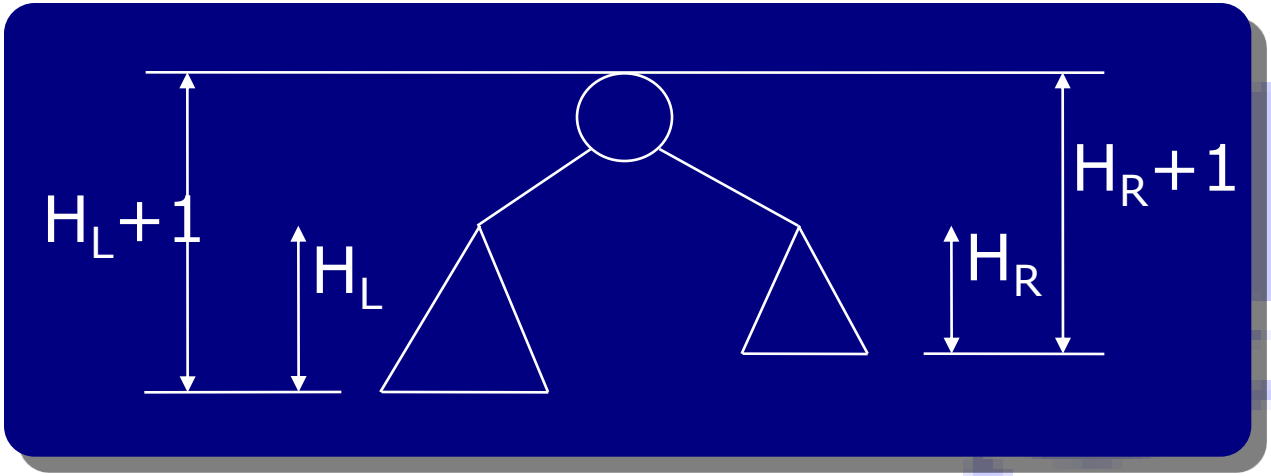
- ADT tree menyimpan referensi dari *root node*, yang merupakan awal untuk mengakses tree.

```
public class BinaryTree<A>
{
    private BinaryNode<A> root;

    public BinaryTree( )
    {
        root = null;
    }
}
```

Berfikir Rekursif

- Menghitung tinggi tree dapat menjadi program yang rumit bila tidak menerapkan rekursif.
- Tinggi sebuah tree adalah: maksimum tinggi dari subtree ditambahkan satu (tinggi dari root).
  - $H_T = \max (H_L+1, H_R+1)$



Menghitung tinggi tree

- Antisipasi *base case* (empty tree).
  - Catatan: Tree dengan hanya satu node memiliki tinggi = 0.
- Terapkan perhitungan/analisa sebelumnya dalam bentuk program.

```
public static int height (BinaryNode<A> t)
{
    if (t == null) {
        return -1;
    } else {
        return max(height (t.left) + 1,
                   height (t.right) + 1);
    }
}
```

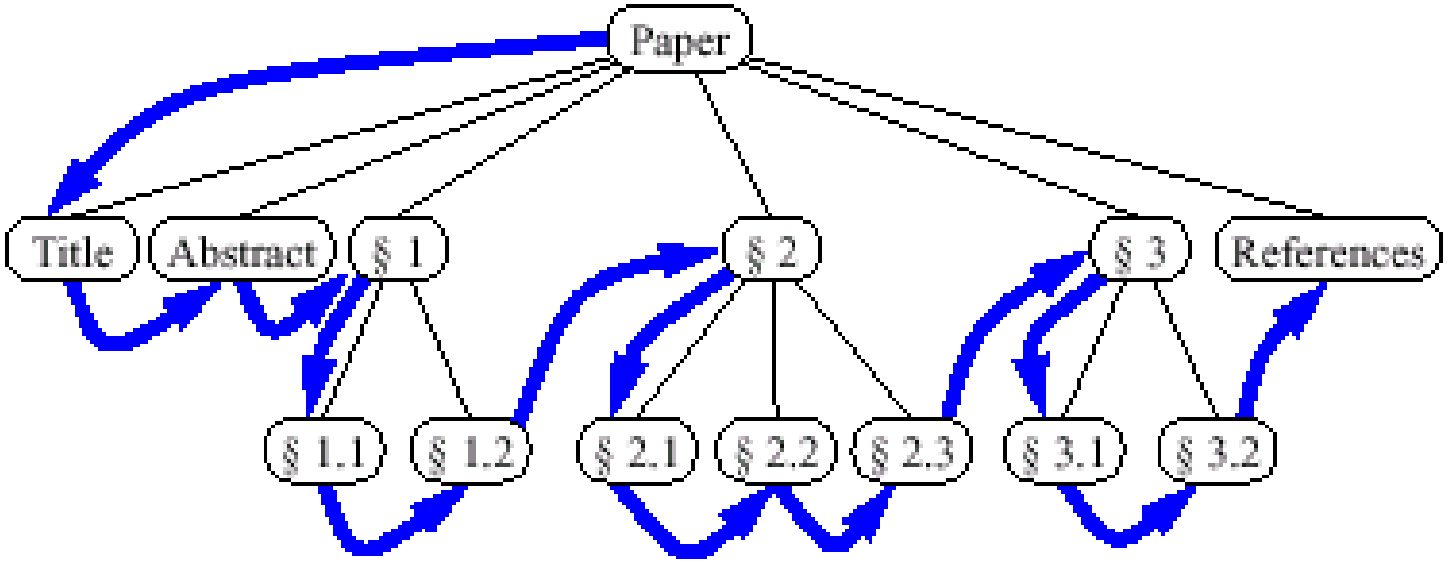
Algoritma pada *Binary Tree*

- Struktur data tree di definisikan / dilihat secara rekursif:
  - sebuah binary tree terdiri dari sebuah node dengan dua buah sub tree (kiri dan kanan) yang masing-masing adalah tree juga.
- Algoritma untuk Binary Tree akan lebih mudah dinyatakan secara rekursif.
- *Binary tree* memiliki dua kasus rekursif
  - *Base case: empty leaf – external node.*
  - *Recursive case*: Sebuah *internal node* (*root*) and dua *binary trees* (*subtree kiri dan subtree kanan*)
- *Traversing Tree*:
  "Menjalani/mengunjungi" *tree*.

Traversing Trees: Preorder traversal

- Contoh: reading a document from beginning to end

```
Algorithm preOrder(v)
        "visit" node v;
        preOrder(leftChild(v));
        preOrder(rightChild(v));
```



Print Pre-Order

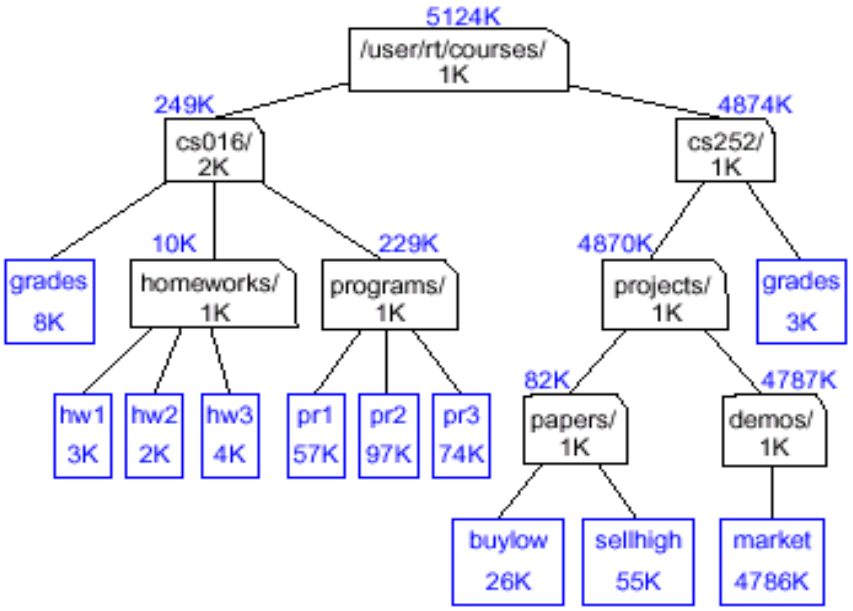
```
class BinaryNode<A> {
      void printPreOrder() {
          System.out.println( element );   // Node
          if( left != null )
               left.printPreOrder( );       // Left
          if( right != null )
               right.printPreOrder( );      // Right
      }
}

public class BinaryTree<A> {
      public void printPreOrder() {
          if( root != null )
               root.printPreOrder( );
      }
}
```

Traversing Trees: Postorder traversal

- Contoh: du (disk usage) command in Unix

```
Algorithm postOrder(v)
    postOrder(leftChild(v));
    postOrder(rightChild(v));
    "visit" node v;
```



Print Post-Order

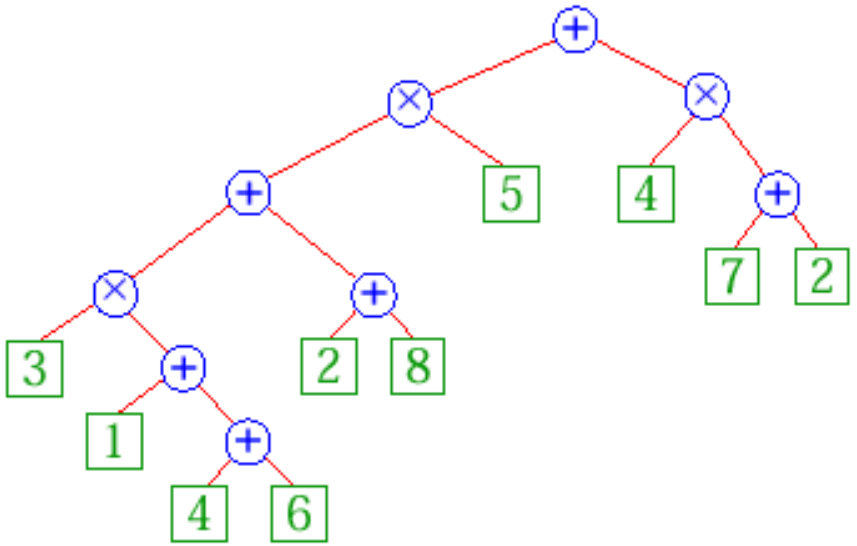
```
class BinaryNode<A> {
       void printPostOrder( )
       {
           if( left != null )
               left.printPostOrder( );          // Left
           if( right != null )
                  right.printPostOrder( );      // Right
           System.out.println( element );       // Node
       }
}

class BinaryTree<A> {
       public void printPostOrder( )
       {
           if( root != null )
              root.printPostOrder( );
       }
}
```

Traversing Trees: Inorder traversal

- Contoh: Representasi ekspresi arithmatik

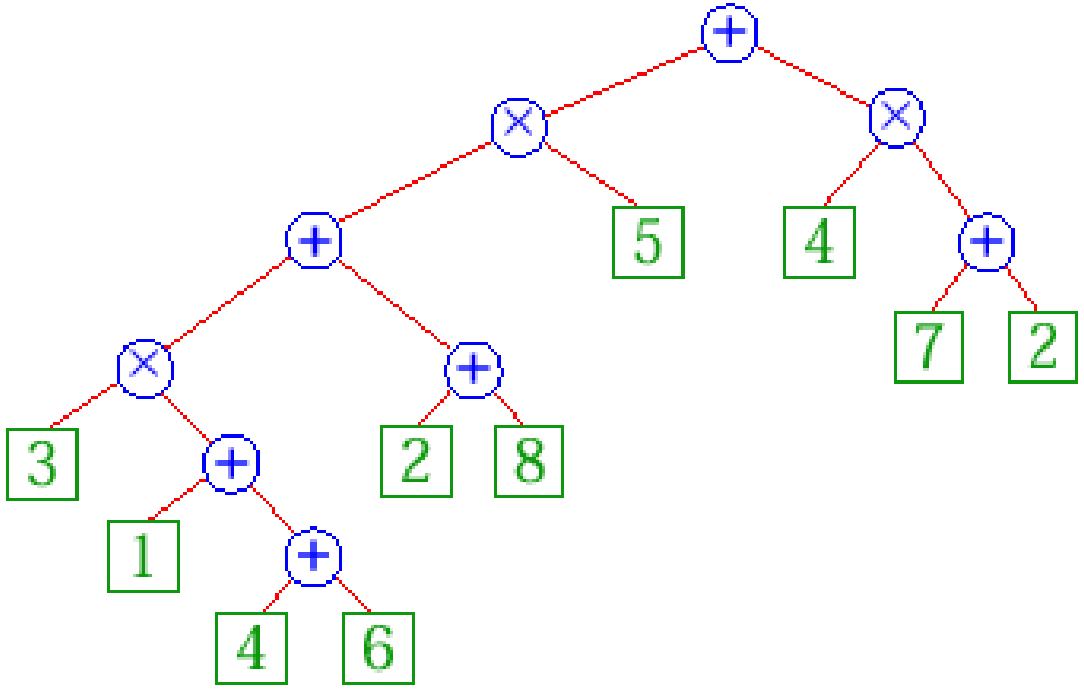
```
Algorithm inOrder(v)
      inOrder(leftChild(v));
      "visit" node v;
      inOrder(rightChild(v));
```



((((3 × (1 + (4 + 6))) + (2 + 8)) × 5) + (4 × (7 + 2)))

Traversing tree: Inorder Traversal

- Contoh: Urutan penulisan ekspresi aritmatika
- Mencetak sebuah expressi aritmatika.
  - print "(" before traversing the left subtree
  - print ")" after traversing the right subtree



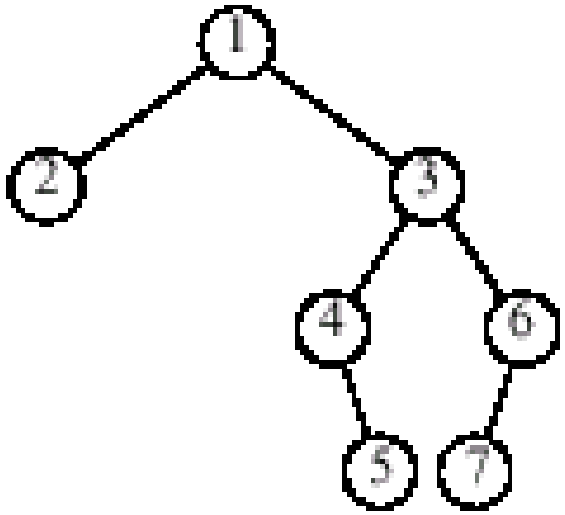
((((3 × (1 + (4 + 6))) + (2 + 8)) × 5) + (4 × (7 + 2)))

Print InOrder

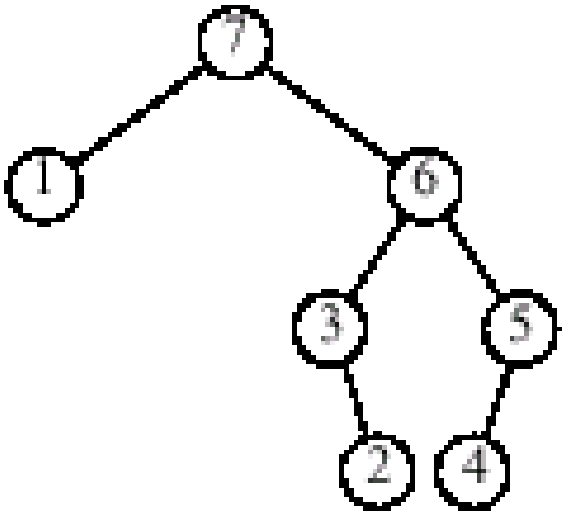
```
class BinaryNode<A> {
       void printInOrder( )
       {
           if( left != null )
               left.printInOrder( );              // Left
           System.out.println( element );         // Node
           if( right != null )
               right.printInOrder( );             // Right
       }
}

class BinaryTree<A> {
       public void printInOrder( )
       {
           if( root != null )
               root.printInOrder( );
       }
}
```

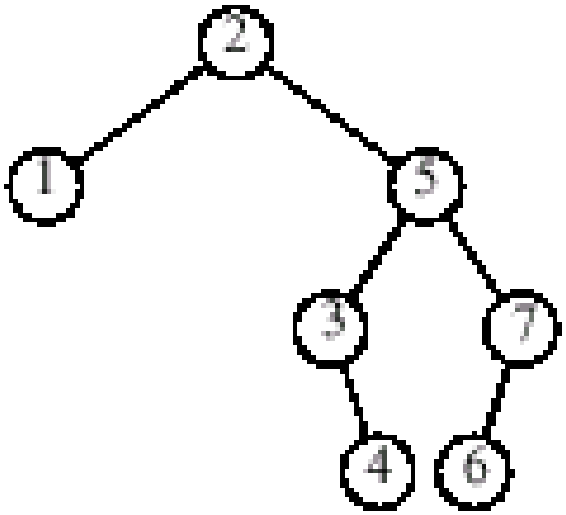
Traversing Tree



Pre-Order



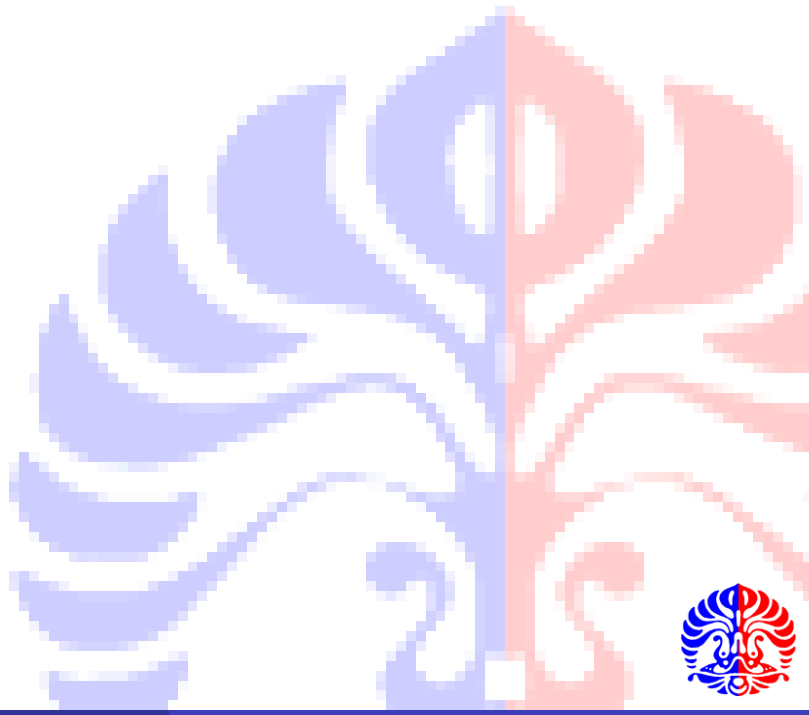
Post-Order



InOrder

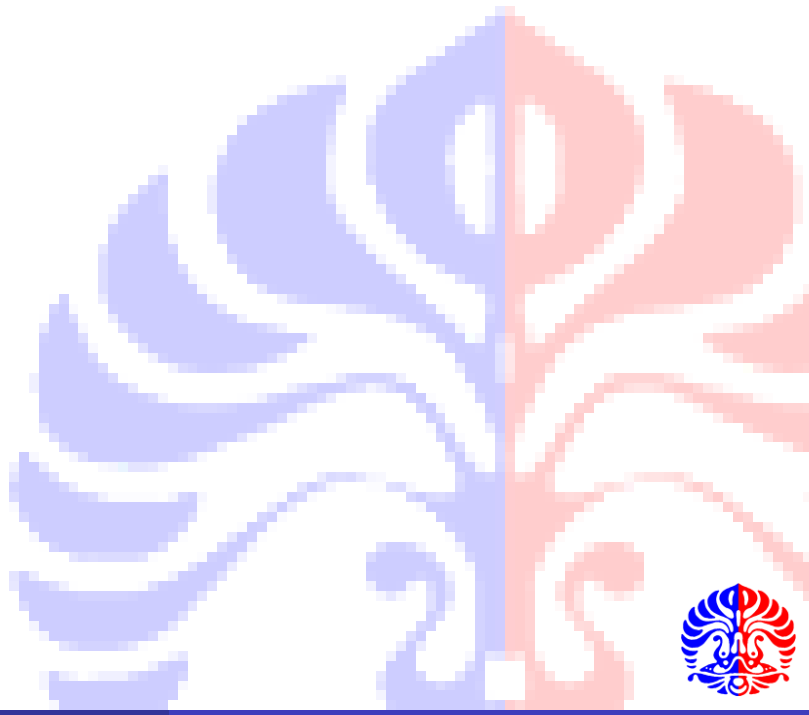
Puzzle

- Sebuah binary tree dicetak dengan menggunakan traversal di bawah ini:
  - Pre-order: 5 23 55 9 42 6 12 14 44
  - In-order: 55 9 23 42 5 14 12 6 44
  - Post-order: 9 55 42 23 14 12 44 6 5
  - Catatan: urutan pencetakan kiri kemudian kanan
- Gambarkan binary tree tersebut!



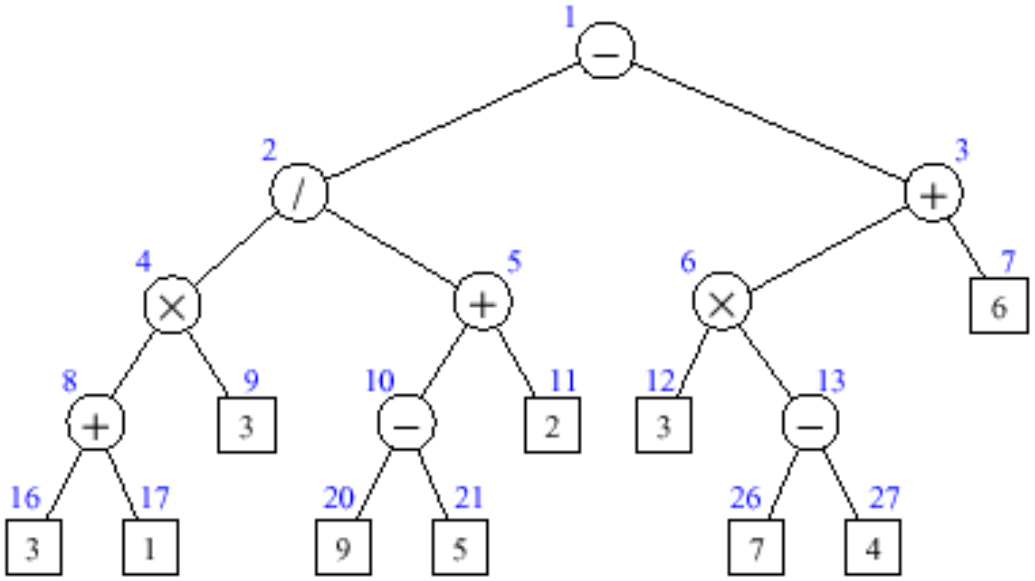
Latihan: Traversing Trees

- Algoritma traversing mana yang sesuai untuk melakukan operasi perhitungan nilai expressi aritmatika yang direpresentasikan menggunakan binary tree?



Jawaban: postorder traversal

```
Algorithm evaluateExpression(v)
    if v is an external node
        return nilai bilangan pada v
    else
        x = evaluateExpression(leftChild(v))
        y = evaluateExpression(rightChild(v))
        // Misalkan o adalah operator pada v
        return x o y
```

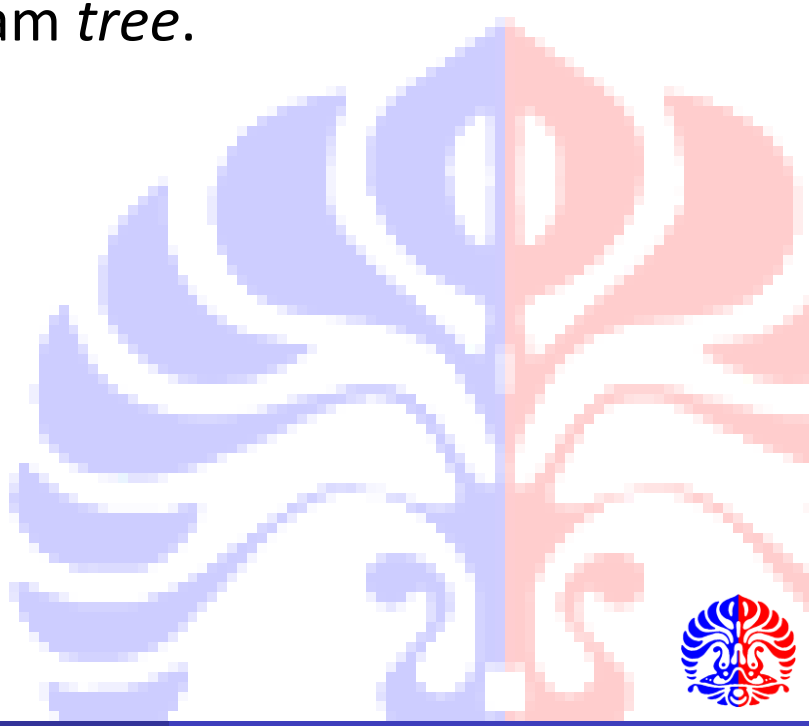


Running Time

- Strategy yang digunakan adalah postorder traversal: information dari *node* dihitung setelah informasi dari seluruh *children* dihitung.
- Postorder traversal *running time* adalah *N* (jumlah elemen dalam *tree*) dikalikan beban waktu untuk memproses tiap *node*.
- *Running time*-nya *linear* karena tiap node hanya diproses sekali dengan beban waktu konstan.

Latihan

- Asumsi: Sebuah binary tree (seluruh *internal node* memiliki degree 2) dengan elemen bilangan bulat.
- Buat algoritma yang melakukan:
  - Pencarian bilangan paling besar.
  - Penghitungan total bilangan dalam *tree*.



Kesulitan dengan rekursif

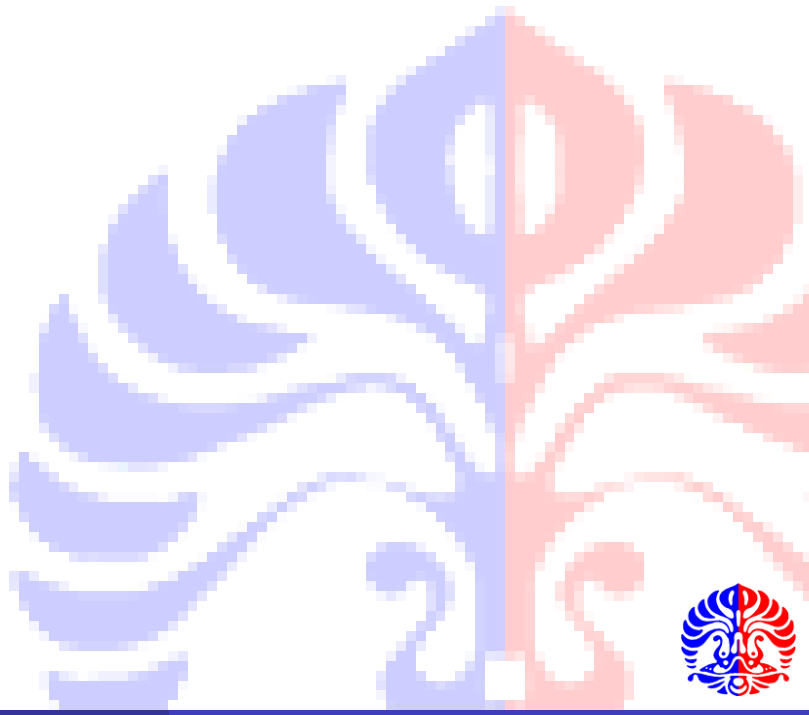
- Terkadang, kita ingin memroses semua node tanpa peduli urutan.
- Agar mudah, kita mau looping sederhana → ***Iterator***
- Bagaimana caranya implementasi secara rekursif?
  - Bayangkan saat current node yang sedang diakses adalah sebuah internal node. Bagaimana menentukan node mana yang akan diakses selanjutnya?
  - Pada fungsi rekursif, informasi ini ***implisit*** pada call stack.
- Bagaimana menghindari rekursif?
- Bagaimana mengimplementasikan traversal yang tidak rekursif?

Rekursif vs. Loop

- Fungsi rekursif dieksekusi Java menggunakan ***stack***.
  - Kita dapat melakukan **traversal non-rekursif** dengan membuat stack sendiri.
  - Dengan kata lain, meng-emulasikan: *stack of activation records*.
- Apakah mungkin non-rekursif lebih cepat dari rekursif?
  - Ya
- Mengapa?
  - Kita dapat menyimpan hanya informasi yang penting saja dalam stack, sementara compiler menyimpan seluruh *activation record*.
  - Namun demikian efisiensi yang dihasilkan tidak akan terlalu besar apalagi dengan teknologi optimisasi *compiler* yang semakin maju.

Tree Iterator: implementation

- Tree iterator dan traversal-nya diimplementasikan secara **non-rekursif** menggunakan **stack**.



Post-Order Traversal dengan Stack

- Gunakan stack untuk menyimpan status terakhir. (node yang sudah dikunjungi tapi belum selesai diproses)
  - sama dengan PC (*program counter*) dalam *activation record*
- Apa saja status pada post-order traversal?

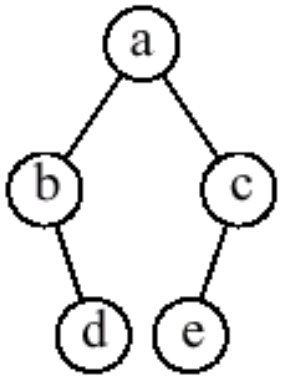
0. akan melakukan rekursif pada subtree kiri
1. akan melakukan rekursif pada subtree kanan
2. akan memproses node yang dikunjungi

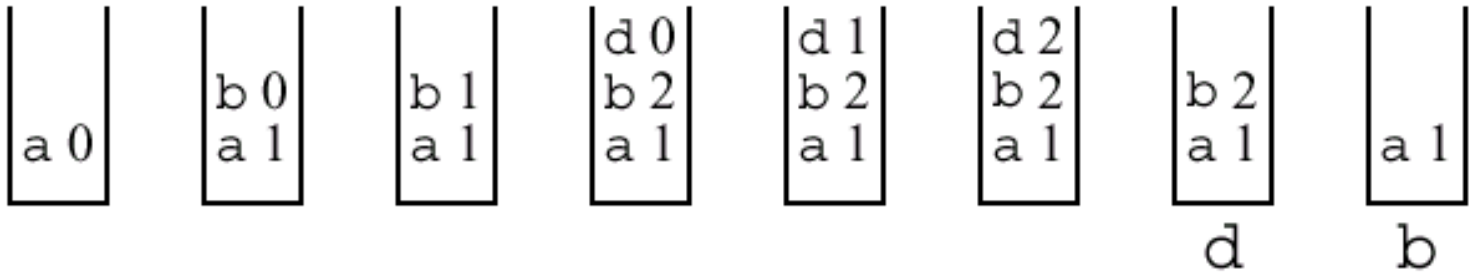
Post-Order Algorithm/Pseudocode

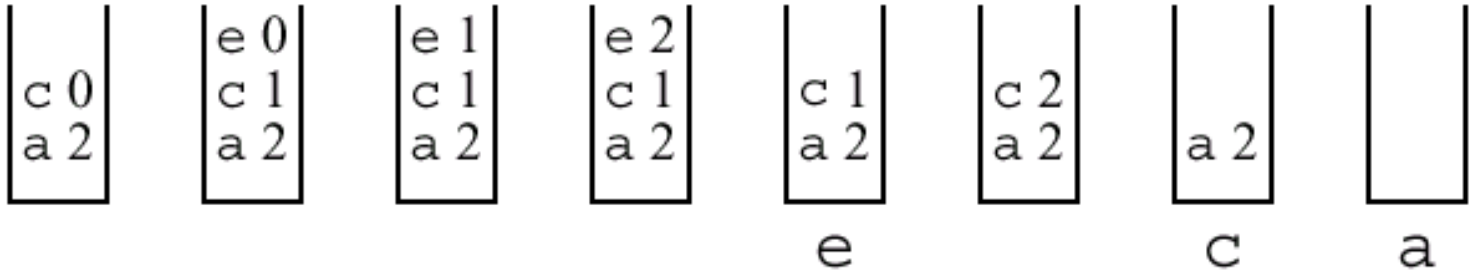
- init: push the root kedalam stack dengan status 0
- advance:

```
while (not stack.empty())
    node X = pop from the stack
    switch (state X):
        case 0:
             push node X with state 1;
             push left child node X (if it exists) w/ state 0;
             break;
        case 1:
             push node X with state 2;
             push right child node X (if it exists) w/ state 0;
             break;
        case 2:
             "visit"/"set current to" the node X;
             return;
```

Post-Order traversal: stack states

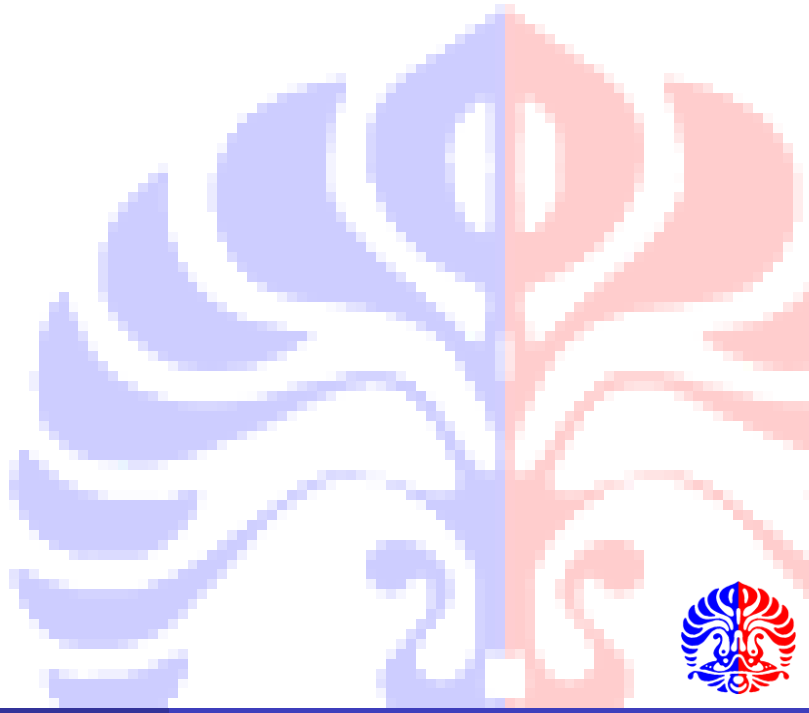






Latihan

- Buat algorithm/pseudo-code dengan in-order traversal menggunakan stack.
- Buat algorithm/pseudo-code dengan pre-order traversal menggunakan stack.



In-Order Traversal using Stack

- Apa saja status pada in-order traversal?

0. akan melakukan rekursif pada subtree kiri

1. akan memproses node yang dikunjungi

2. akan melakukan rekursif pada subtree kanan

In-Order Algorithm/Pseudocode

- init: push the root into the stack with state 0
- advance:

```
while (not stack.empty())
    node X = pop from the stack
    switch (state X):
        case 0:
             push node X with state 1;
             push left child node X (if it exists) w/ state 0;
             break;
        case 1:
             push node X with state 2;
             "visit"/"set current to" the node X;
             return;
        case 2:
             push right child node X (if it exists) w/ state 0;
             break;
```

In-Order Algorithm/Pseudocode

- init: push the root into the stack with state 0
- advance (optimize):

```
while (not stack.empty())
    node X = pop from the stack
    switch (state X):
        case 0:
             push node X with state 1;
             push left child node X (if it exists) w/ state 0;
             break;
        case 1:
             "visit"/"set current to" the node X;
             push right child node X (if it exists) w/ state 0;
             return;
```

Pre-Order Traversal using Stack

- Apa saja status pada pre-order traversal?

0. akan memproses node yang dikunjungi

1. akan melakukan rekursif pada subtree kiri

2. akan melakukan rekursif pada subtree kanan

Pre-Order Algorithm/Pseudocode

- init: push the root into the stack with state 0
- advance:

```
while (not stack.empty())
    node X = pop from the stack
    switch (state X):
        case 0:
             "visit"/"set current to" the node X;
             push node X with state 1;
             return;
        case 1:
             push right child node X (if it exists) w/ state 0;
             push node X with state 2;
             break;
        case 2:
             push left child node X (if it exists) w/ state 0;
             break;
```

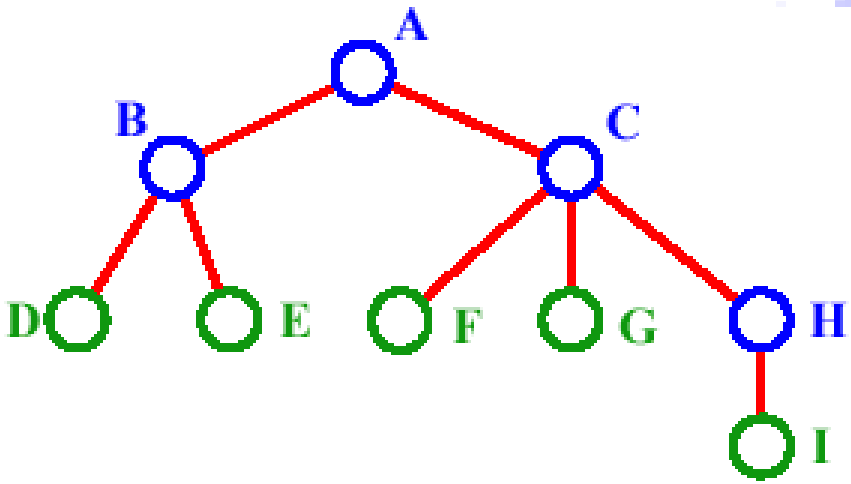
Pre-Order Algorithm/Pseudocode

- init: push the root into the stack
- advance (optimized):

```
if (not stack.empty())
    node X = pop from the stack
    "visit"/"set current to" the node X;
    push right child node X (if it exists);
    push left child node X (if it exists);
```

Level-order Traversal

- Kunjungi root diikuti oleh seluruh node pada sub tree dari kiri ke kanan kemudian diikuti oleh node pada sub tree-nya lagi.
- Tree dikunjungi berdasarkan level.
- Pada tree dibawah urutan kunjungan adalah: A - B - C - D - E - F - G - H - I



Level-order Traversal: idea

- Gunakan queue bukan stack
- Algorithm (mirip dengan pre-order)
  - init: enqueue the root into the queue
  - advance:

```
node X = dequeue from the queue
“visit”/”set current to” the node X;
enqueue left child node X (if it exists);
enqueue right child node X (if it exists);
```

Latihan

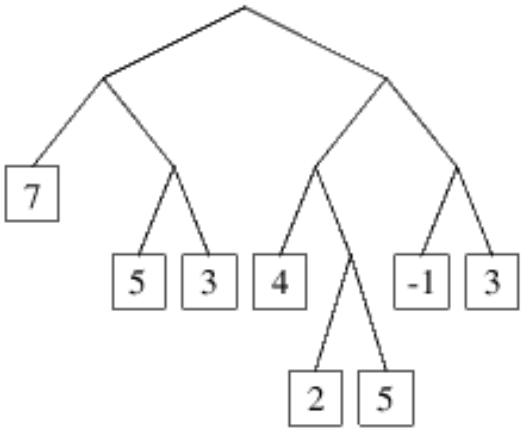
- Buat program untuk mencetak isi dari sebuah binary tree secara level order
  - Implementasikan menggunakan queue
- Dapatkah anda membuat implementasi yang lebih mudah / sederhana?
  - Hint: Coba pikirkan representasi binary tree yang lain, sehingga implementasi level order dapat menjadi lebih sederhana.

Representasi binary tree dengan array

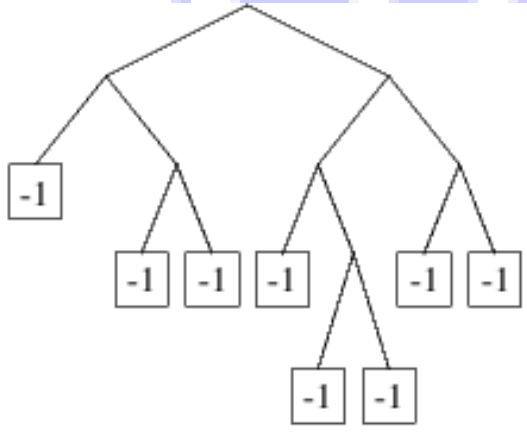
- Complete binary tree dengan *n* nodes dapat diresentasikan menggunakan array dengan index dari 1..n
- Untuk setiap node dengan index *i*, maka:
  - Parent (*i*) terletak pada index $\lfloor i/2 \rfloor$ if $i \neq 1$; for $i = 1$, tidak ada parent.
  - Left-child (*i*) terletak pada $2i$ if $2i \leq n$. (else tidak ada left-child)
  - Right-child (*i*) terletak pada $2i+1$ if $2i + 1 \leq n$ (else tidak ada right-child)
- Akan dianalisa lebih dalam pada materi Heap.

Latihan

- Pada ADT tree yang berisikan elemen bilangan bulat. Hitung elemen paling kecil pada *leaves* dan update seluruh *leaves* pada *tree* tersebut dengan elemen terkecil tersebut. (*Repmin problem*)



Input



Output