Data Structures and Algorithms (資料結構與演算法)

Lecture 9: Binary Tree

Hsuan-Tien Lin (林軒田)

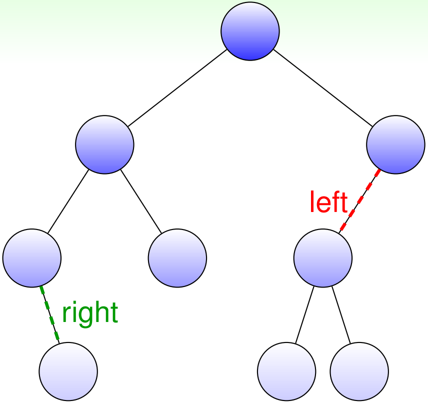
htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

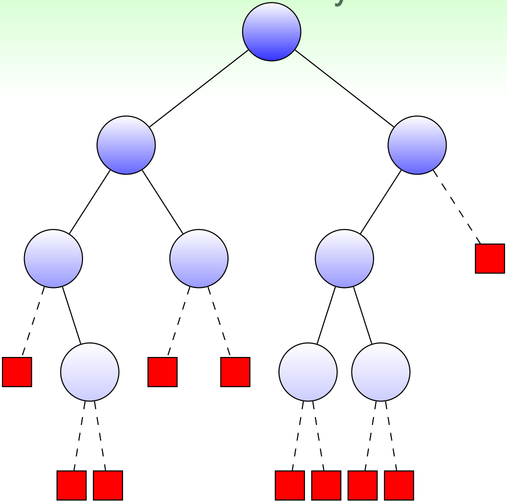
terminologies

Binary Tree



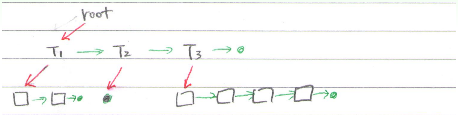
binary tree: rooted tree with node degree ≤ 2
& left/right difference

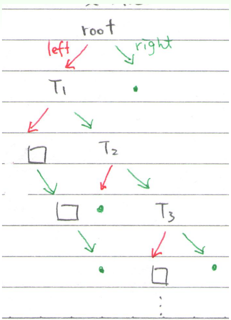
Extended Binary Tree



extended binary tree: added empty external nodes
so all degree = 2 & all original nodes internal

General Tree with Linked Lists ≡ Binary Tree



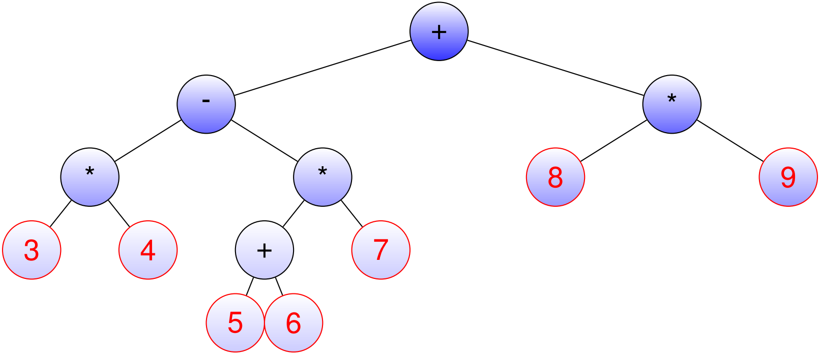


left-child right-sibling implementation of general tree

binary tree traversals

Expression Tree

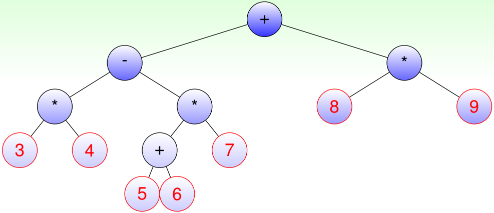
$$3 * 4 - (5 + 6) * 7 + 8 * 9$$



expression tree: (binary) expression
represented by a (binary) tree

Outputting Infix Expression from Expression Tree

want: $3*4-(5+6)*7+8*9$



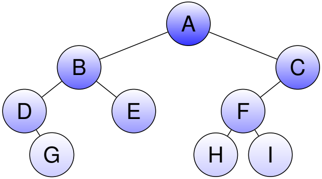
```
Output-Infix(T)

if Is-Leaf(T)
        print T.data
else
        if [T.data higher precedence than T.left.data] print '('
        Output-Infix(T.left)
        if [T.data higher precedence than T.left.data] print ')'
        print T.data
        if [T.data higher precedence than T.right.data] print '('
        Output-Infix(T.right)
        if [T.data higher precedence than T.right.data] print ')'
```

Output-Infix: variant of Inorder-Traversal

From Output-Infix to Inorder-Traversal

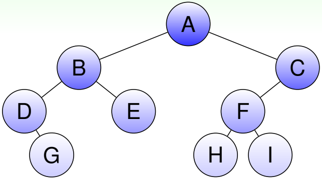
```
Output-Infix(T)
if Is-Leaf(T)
    print T.data
else
    if ...
    Output-Infix(T.left)
    if ...
    print T.data
    if ...
    Output-Infix(T.right)
    if ...
```

```
Inorder-Traversal(T)
if T is NIL

else

    Inorder-Traversal(T.left)

    action with T.data

    Inorder-Traversal(T.right)

```



Inorder-Traversal visit sequence: DGBEAHFIC

POSTORDER-TRAVERSAL & PREORDER-TRAVERSAL



GDEBHIFCA

```
POSTORDER-TRAVERSAL(T)
if T is not NIL
    POSTORDER-TRAVERSAL(T.left)
    POSTORDER-TRAVERSAL(T.right)
    action with T.data
// appl: evaluate expression tree
```

DGBEAHFIC

```
INORDER-TRAVERSAL(T)
if T is not NIL
    INORDER-TRAVERSAL(T.left)
    action with T.data
    INORDER-TRAVERSAL(T.right)
// appl.: output infix expression
```

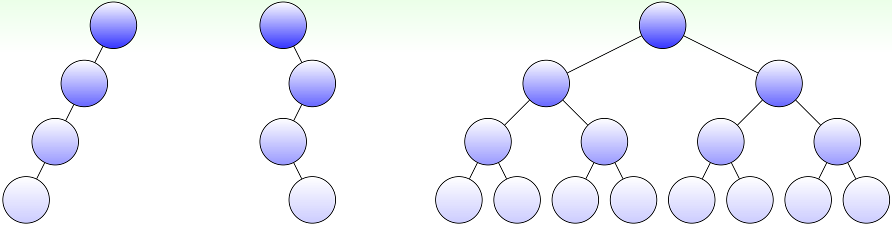
ABDGECFHI

```
PREORDER-TRAVERSAL(T)
if T is not NIL
    action with T.data
    PREORDER-TRAVERSAL(T.left)
    PREORDER-TRAVERSAL(T.right)
// appl.: compare two trees
```

traversal: template for designing tree algorithms

full and complete binary trees

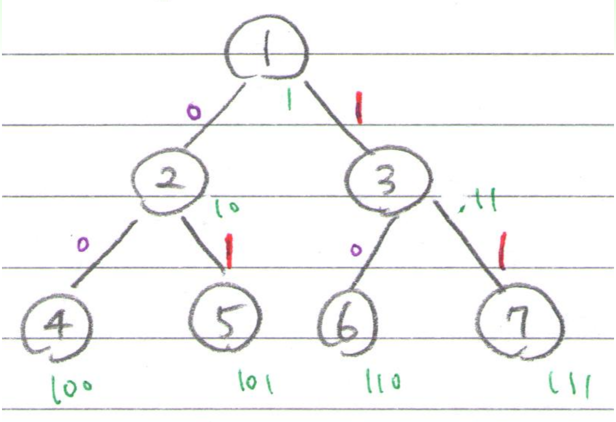
# Nodes in Binary Trees



height	min	max
1	1	1
2	2	3
3		
…		
h	h	$2^h - 1$
	(skewed)	full

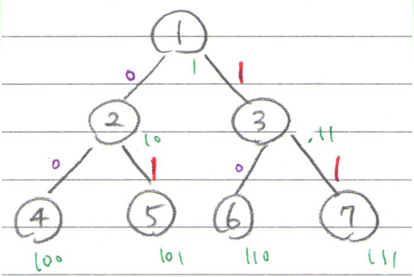
$$h \leq n \leq 2^h - 1 \Leftrightarrow \lg(n+1) \leq h \leq n$$

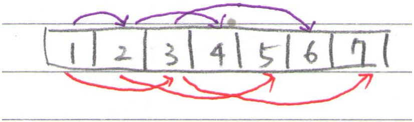
Node Index in Full Binary Tree



node index $= (1\textit{path code})_2$

Representing/Packing Full Binary Tree in Array

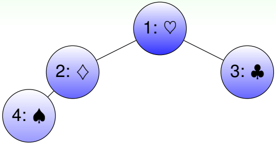




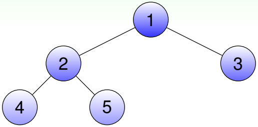
- implicit links: no need for explicit pointers
- can similarly pack any binary tree if NIL can represent NO-DATA (with **space wasting** in data field)

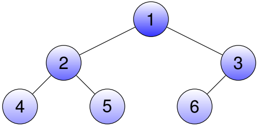
> complete binary tree: full binary tree with first *n* nodes
> (**no waste** with array representation)

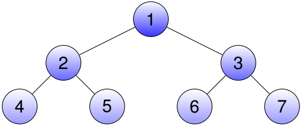
Complete Binary Trees



1	2	3	4
♡	♢	♣	♠







will use this **tree-in-array** property next time

Summary

Lecture 9: Binary Tree

- terminologies

  binary tree $\equiv$ tree with left/right sub-trees
- binary tree traversals

  templates for (recursive) tree algorithm design
- full and complete binary trees

  can be implemented in array without wasting space