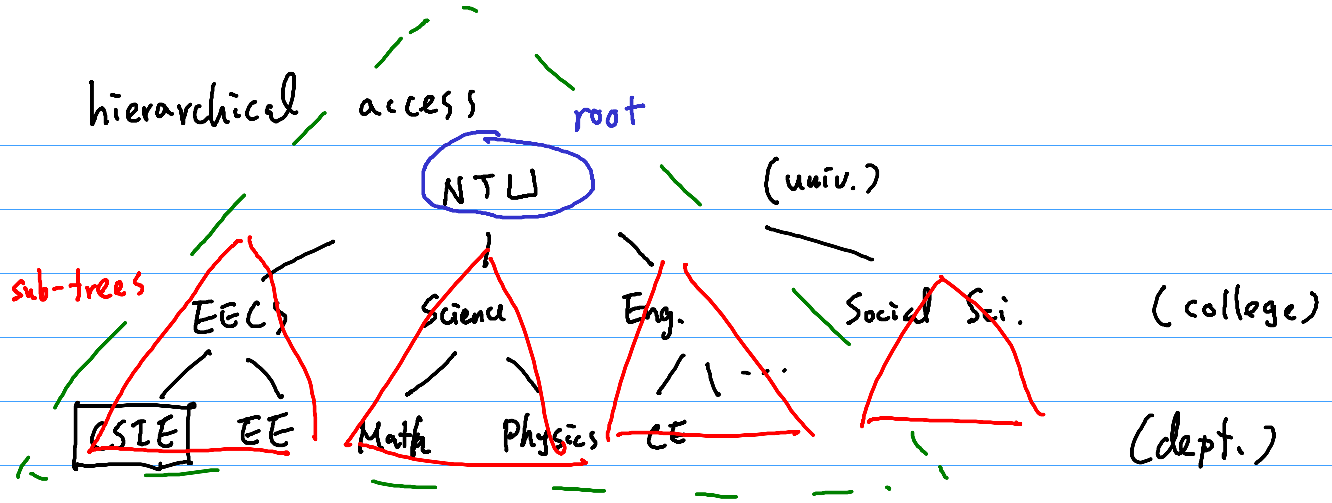
Tree : hierarchical access



* dir / subdir / subdir / file

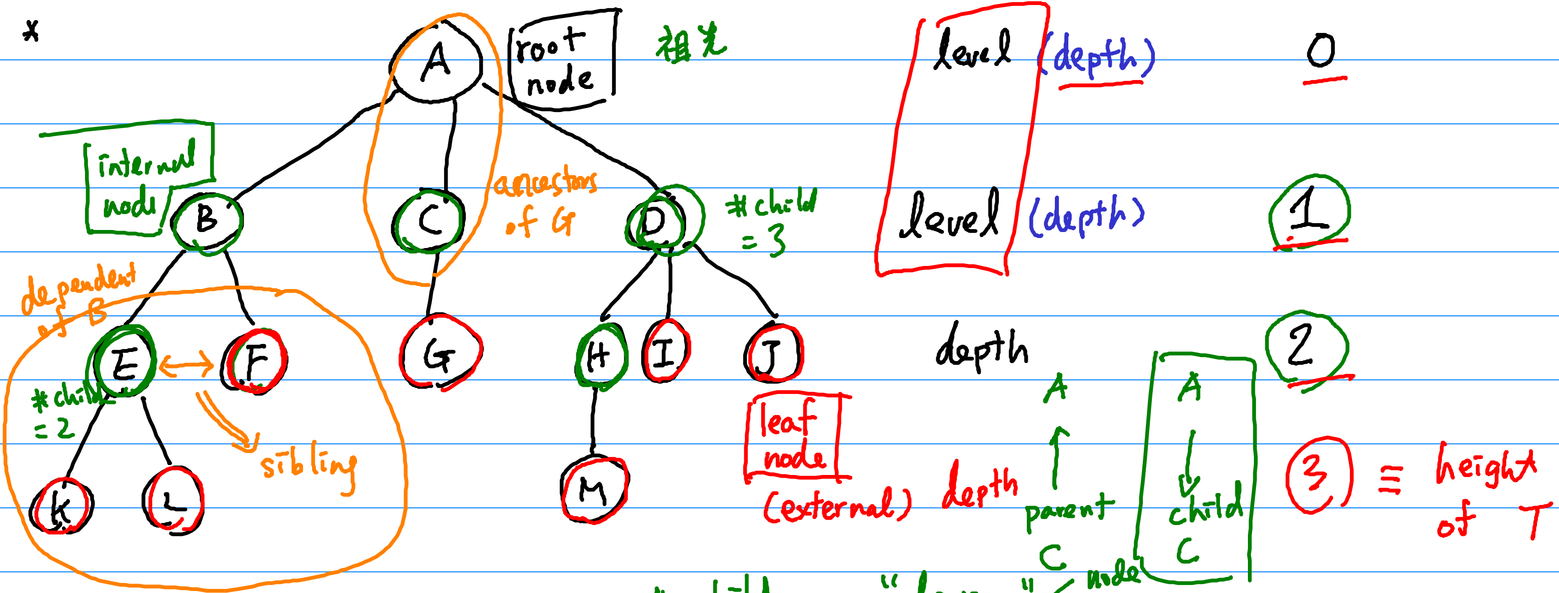
ordered / unordered

* $T \equiv (\text{root} ; T_1 , T_2 , \cdots , T_n)$ recursive definition

$T_1 \equiv (\text{sub-root} ; T_{11} , T_{12} , T_{13} \cdots , T_{1m})$ termination condition

$T_{term} \equiv (\text{root} ; \times)$

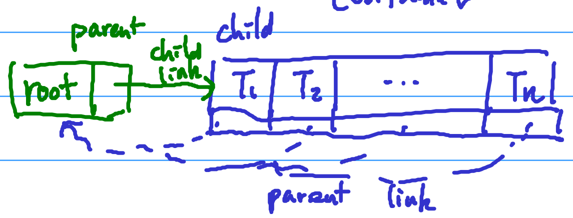
*



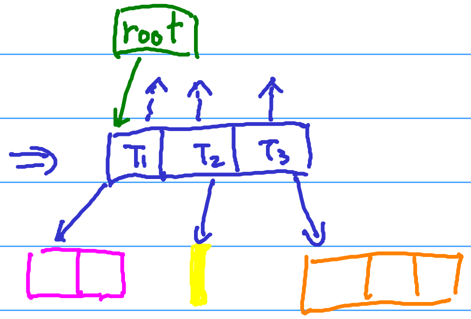
max # child "degree" < node / tree (max node degree)

* represent

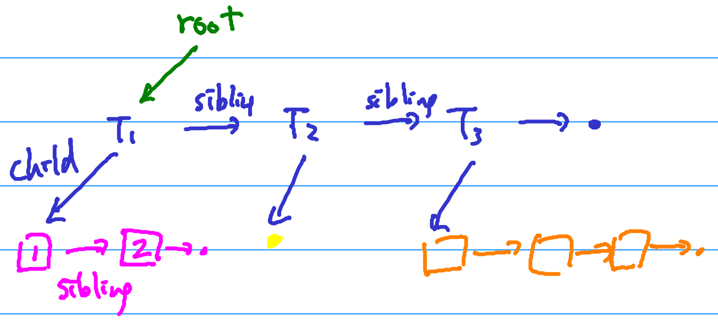
$$T \equiv (\boxed{root}; \underbrace{\boxed{T_1, T_2, \cdots, T_n}}_{container})$$



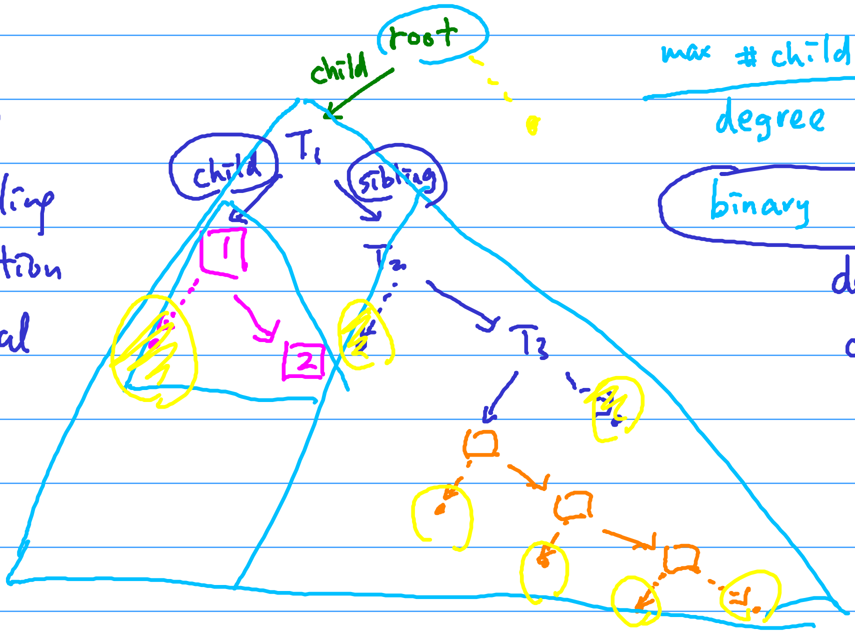
* use vector as container



* use l-list as container



left-child right-sibling representation of general tree



max # child = 2

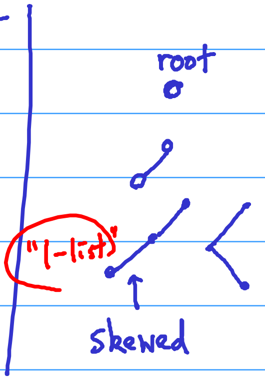
degree = 2

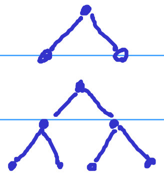
binary tree

deg 2
=
ordered

```
func constructTree(data)
if data empty
    return null
else
    create root node from data[0]
    partition data to n sets
    for i = 1 to n
        T[i] = constructTree(i-th partition of data)
    link root to T[i]'s
    return address of root node
```

* how many nodes ?





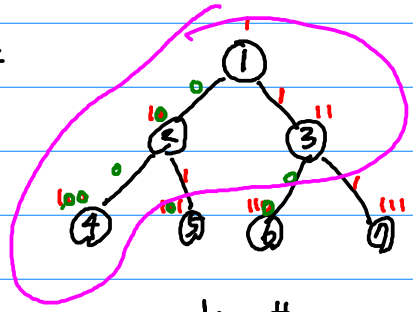
# node

$h = 0$	min = 1	max = 1
$h = 1$	min = 2	max = 3
$h = 2$	min = 3	max = 7
	⋮	
	min = $h+1$	max = $2^{h+1} - 1$

$$h+1 \le n \le 2^{h+1} - 1$$

$$\overset{\text{好}}{\underset{\log_2}{\lg(n+1) - 1}} \le \boxed{h} \le \overset{\text{不好}}{n-1}$$

*

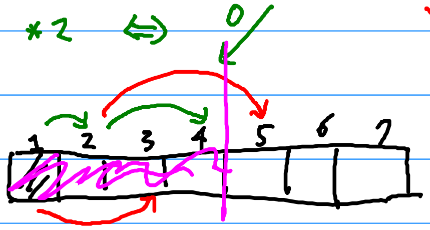


full binary tree   complete binary tree

"binary representation" of node #   1, 2, ..., n

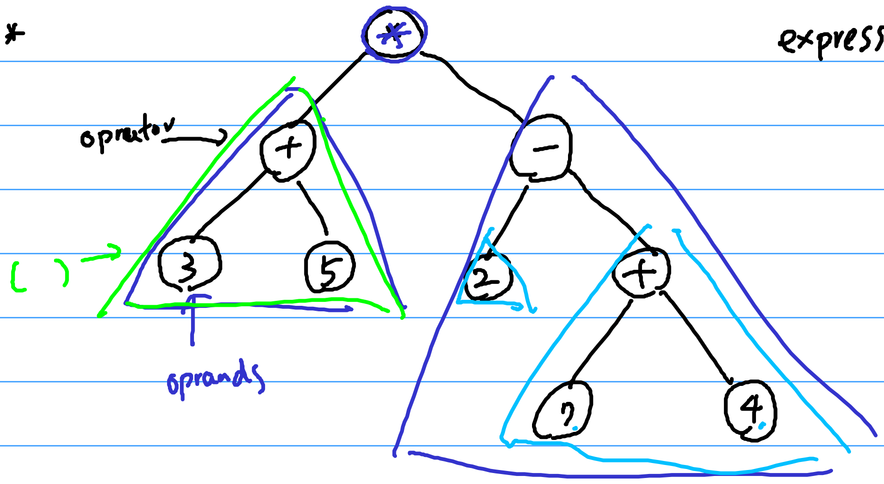
$$\text{node } \# = (1\ (\text{path code}))_2$$

$*2 \Leftrightarrow 0 \swarrow$ $\searrow 1 \Leftrightarrow *2+1$



~~link~~

*



expression tree

root

$$\underbrace{(3+5)}_{\text{sub-tree}} \ *\ (2 - (7+4))$$