Lecture 15

binary trees

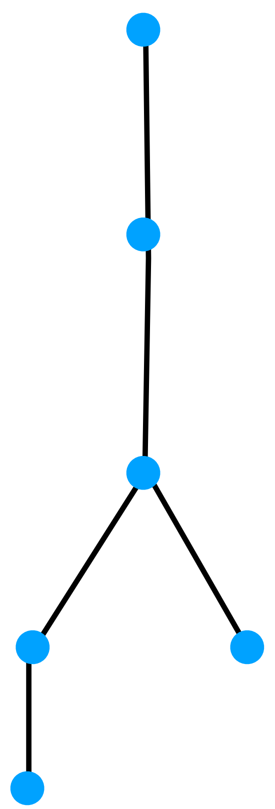
Read Schaum sections: 8.8, 9.4, 10.1-3, 10.5, 10.9

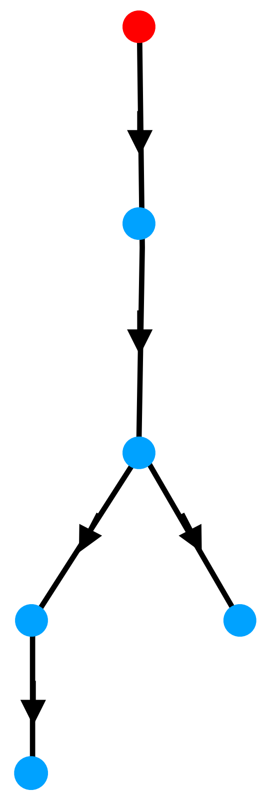
Recap

Tree: acyclic connected graph (undirected, directed, rooted, ordered)

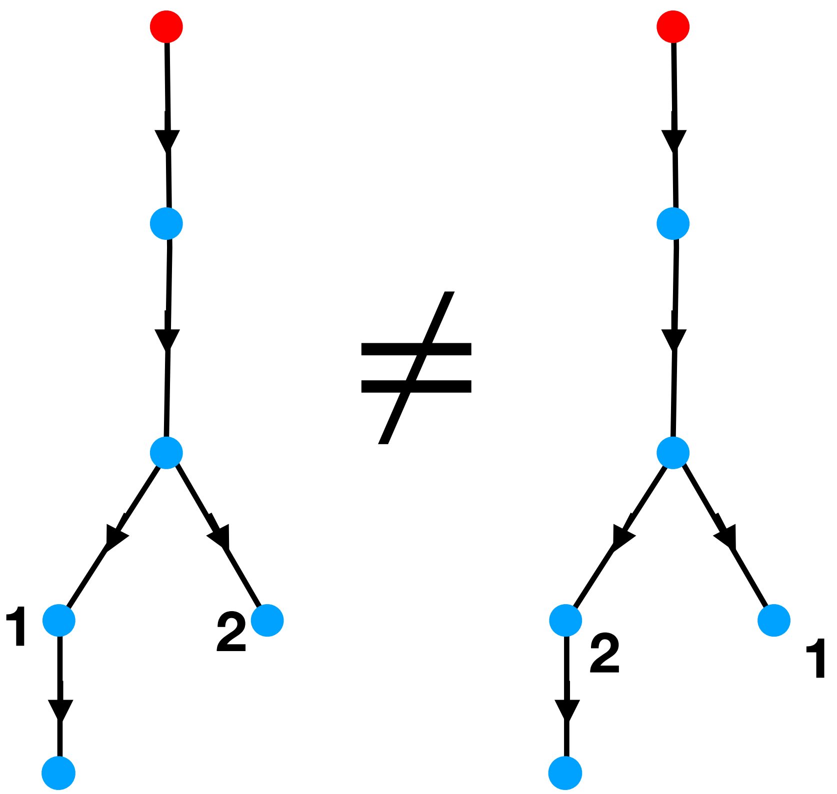
Rooted: there is a special element (formally: $T = (V, E, r),\ r \in V$

Ordered: children are *ordered*





implicit directionality



isomorphic as graphs, but not as ordered graphs…

Main properties of trees

Tree T
(=acyclic& connected)

$\Leftrightarrow$

$\exists$ *unique*
simple path between
any two vertices

$\Rightarrow$

T-{e} is not connected
(two trees)

Further:

***Trees* have *n-1* edges.**

They are *minimally connected*, *maximally acyclic*.

Properties of trees

recall: tree is an undirected acyclic graph

Lemma. Let G=(V,E) be an [undirected] tree. Then $\|E\| = \|V\|-1$

Proof 1: *induction over the number of vertices.*

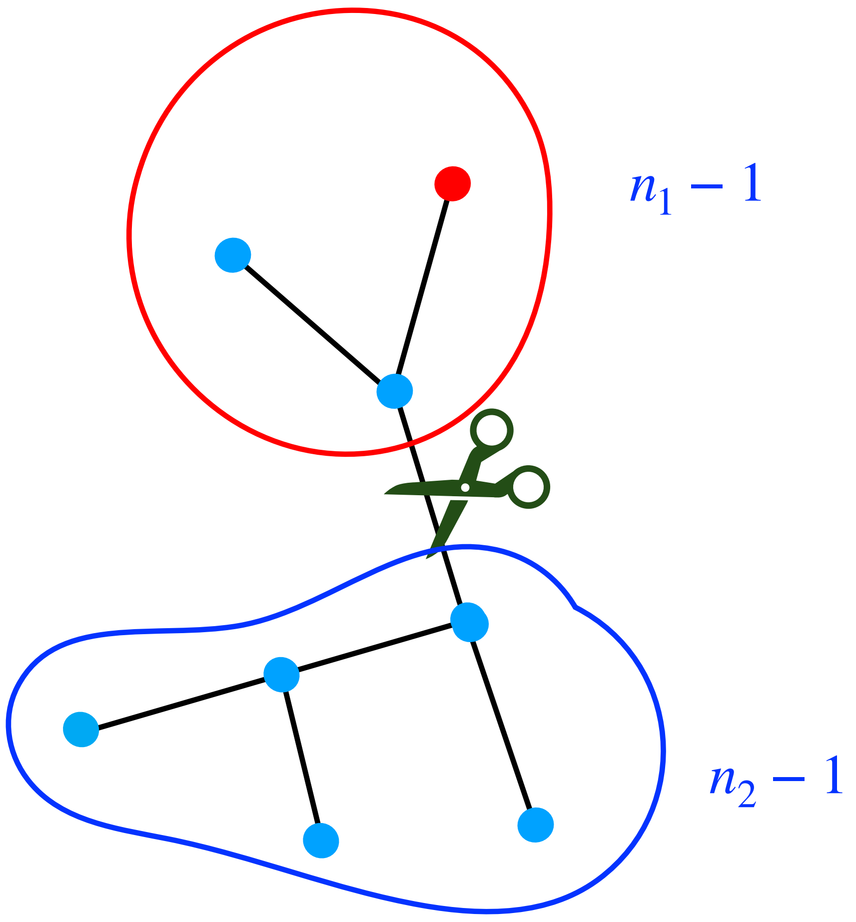
(i) basis: n=1, works.

(ii) assume holds for all k<n.

take any tree of n vertices, and cut any edge.
Now we have two trees (see last slide) with $n_1, n_2, n_1 + n_2 = n$ vertices.

By inductive hypothesis, they have
$n_1 - 1 + n_2 - 1 = n - 2$ *edges in total.*

Since you cut one edge, the initial graph must have had n-1 edges



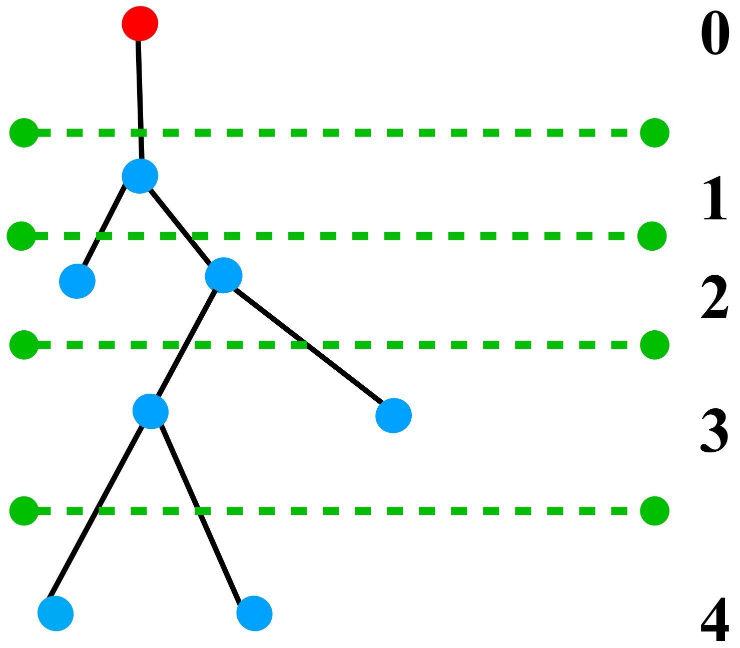
basis

Properties of trees

Lemma. Let G=(V,E) be an [undirected] tree. Then |E| = |V|−1

Proof 2:

Choose a root and look at levels of rooted tree.



Each vertes has exactly one prececessor
in previous level…except the root.
So n-1 edges.

Properties of trees

Theorem (Characterization of trees).

For a graph G (over n vertices) the following are equivalent

(1) G is a tree (connected acyclic graph)
(2) G is maximally acyclic : adding an edge to G creates a cycle
(3) G is minimally connected: removing any edge makes it unconnected
(4) G is acyclic and has n-1 edges
(5) G is connected and has n-1 edges

Use?

NB: this characterization is sometimes given in two parts: 1-2-3, and 1-4-5. See slides of previous lectures.

Properties of trees

Theorem (Characterization of trees).

For a graph G (over n vertices) the following are equivalent

(1) G is a tree
(2) G is maximally acyclic : adding an edge to G creates a cycle
(3) G is minimally connected: removing any edge makes it unconnected
(4) G is acyclic and has n-1 edges
(5) G is connected and has n-1 edges

Proof:

$(1) \Rightarrow (2)$: a) is acyclic ✓
b) adding edge makes a cycle :
add new $e = \{v, w\}$; But G was connected
$\Rightarrow$ $\exists$ simple path $v, v_1 .. w$. (and $\{v,w\}$ is not in)
so $v, v_1 .. w, v$ is a cycle.

$(2) \Rightarrow (1)$ (2) $\Rightarrow$ - connected. otherwise connecting two unconnected components doesn't make a cycle
- connected + acyclic $\Rightarrow$ tree per definition

$(1) \Rightarrow (3)$ tree $\Rightarrow$ connected. & acyclic.
$\Rightarrow$ removing edge disconnects (or cycle!) [seen before]
$\Rightarrow$ minimally connected

$(3) \Rightarrow (1)$ minimally connected $\Rightarrow$ connected
need acyclic.
assume connected & cyclic $\Rightarrow$ cut cycle
$\Rightarrow$ not disconnected. $\Rightarrow$ acyclic. tree.

Proof (continued):

(1) => (4) acyclic by definition. n-1 edges property proven before

(4) => (5) acyclic + (n-1) edges => connected

Assume not: k components, all acyclic => k trees

=> $n_1 + n_2 .. n_k = n$ & $n_1 - 1 + n_2 - 1 + .. n_k - 1 = n - 1$

=> $n - k = n - 1$ => $k = 1$. 1 tree => connected

(5) => (1)

Assume connected, n-1 edges & cycle

=> can remove edge.

=> Connected graph over n-vertices with n-2 edges (see 8.39)

So not a tree but connected => has simple cycle.

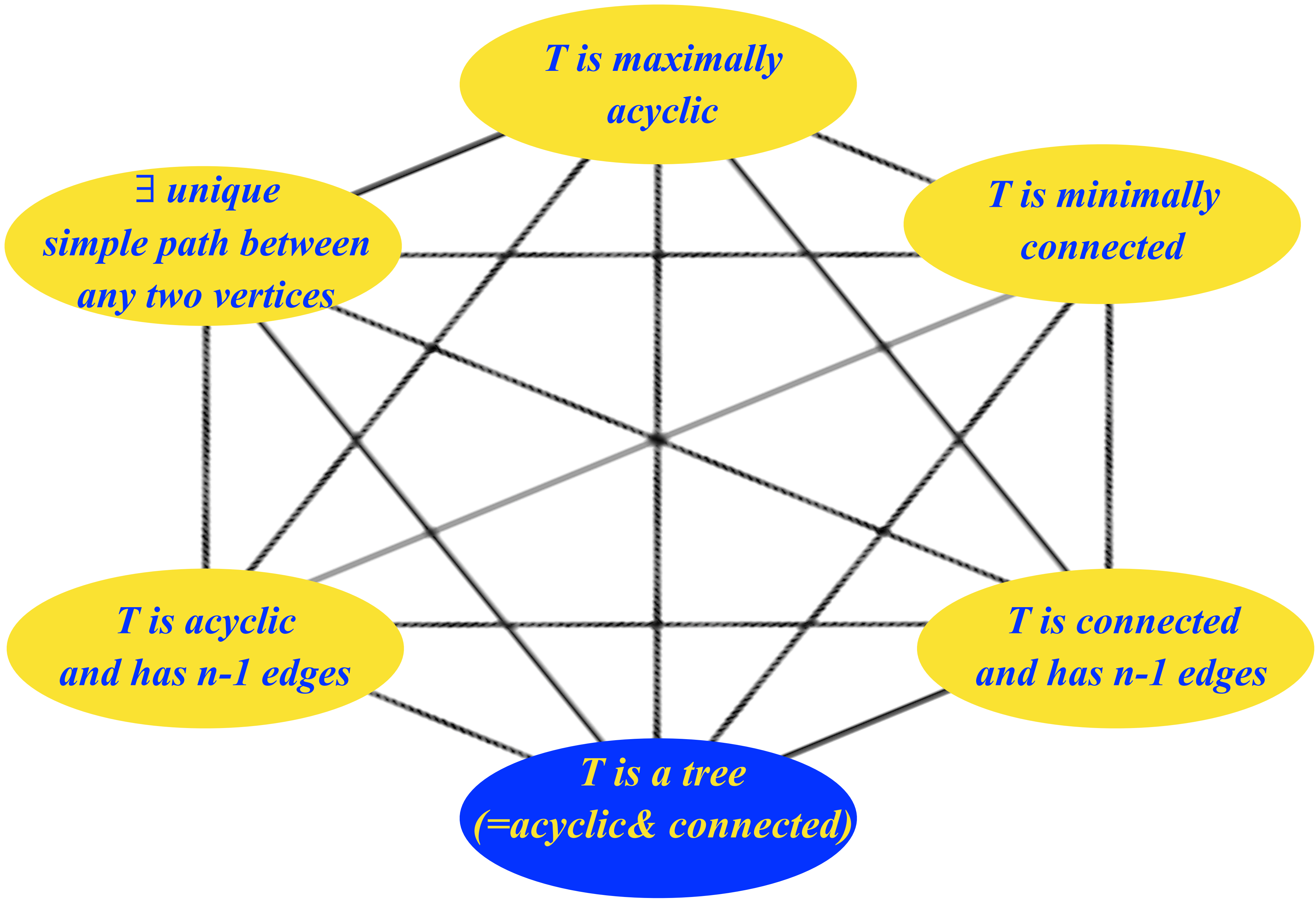
Can remove one edge, so connected with n-3 edges

... => not a tree, has cycle, ... n-4

=> repeat until empty graph. (inductive)

OR: Has spanning tree! ↯

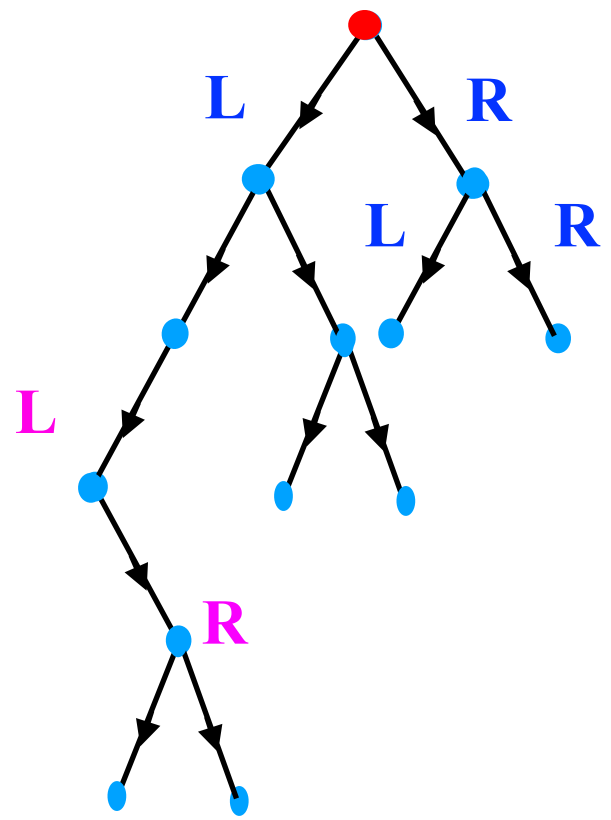
Main properties of trees



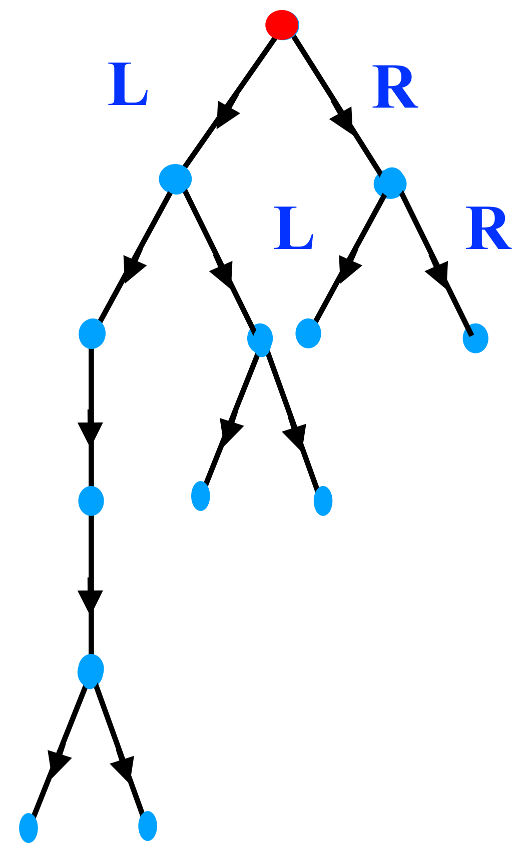
Each edge is a bi-implication

Binary trees

Def. A binary tree is a rooted ordered tree, where each vertex has at most 2 children. The ordering assigns the label "left" and "right" to each child, *even if the child is a single child.*



binary tree



Rooted ordered tree of outdegree <3

In CS binary trees are special

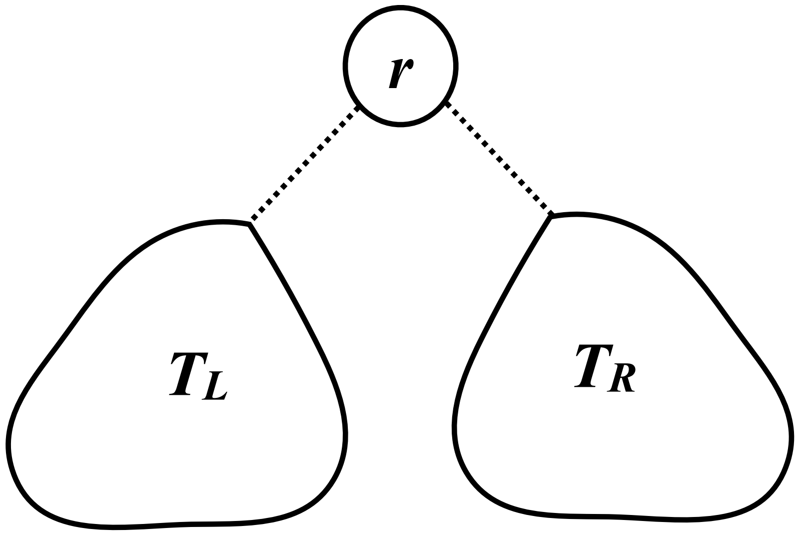
Def. (recursive) A binary tree *T* is a finite set of elements (vertices), such that it is

1) ***T* is empty, or**
2) ***T* contains a distinguished node *R*, called the root of *T* , and the remaining nodes of T form an *ordered pair* of disjoint binary trees T_L and T_R.**

Def. Binary tree (recursive) simplified

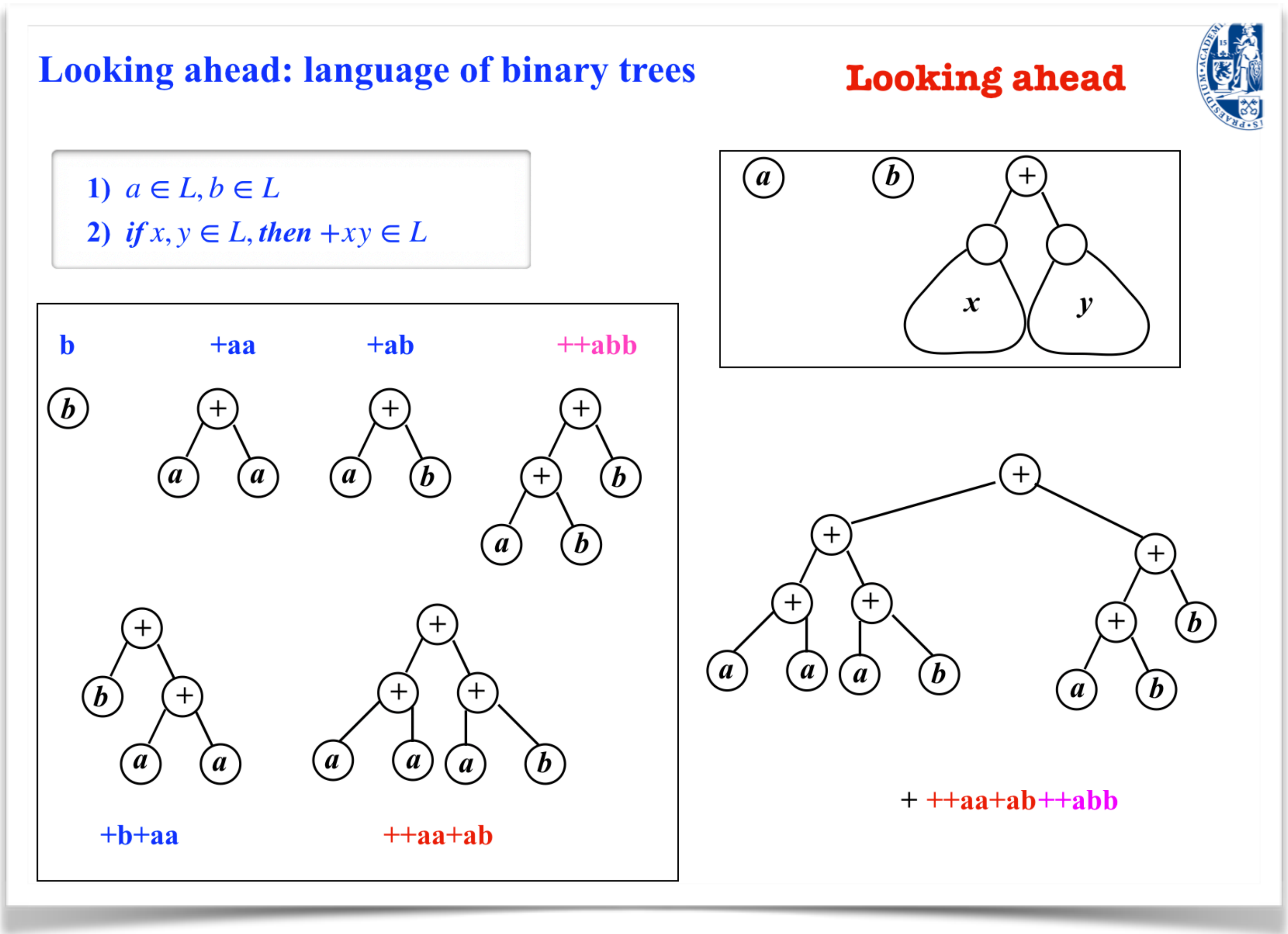
1) ***empty***
2) ***or has a root with a left and right subtree (each is a tree)***

empty **OR**



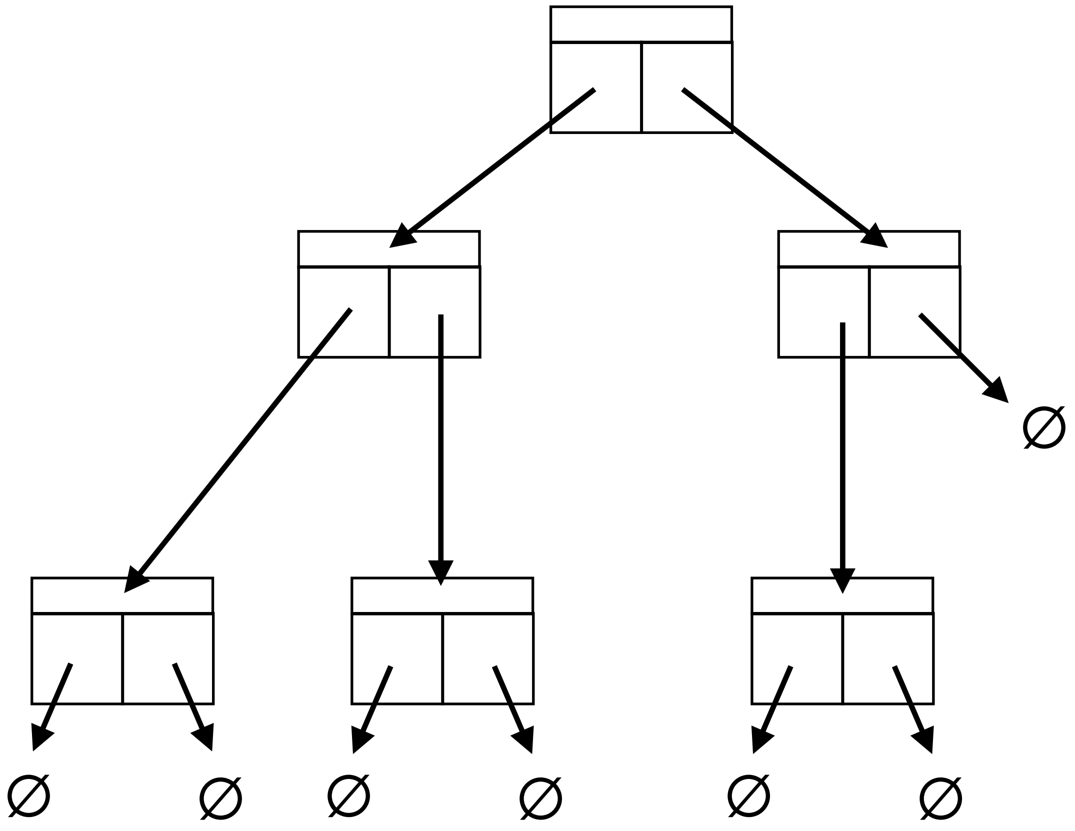
NB. subtrees can be empty

FLASHBACK


Looking ahead: language of binary trees
Looking ahead
1) $a \in L, b \in L$
2) *if* $x, y \in L$, *then* $+xy \in L$
b
+aa
+ab
++abb
+b+aa
++aa+ab
+ ++aa+ab++abb

In CS binary trees are special

a *pointer* structure:



Types of binary trees

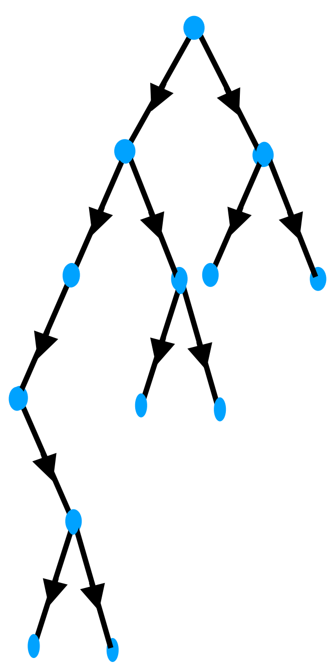
-complete:

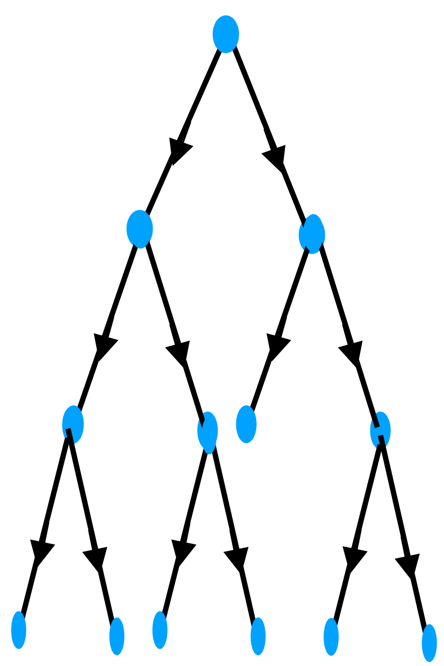
(1) **every level, *except possibly the last*, is completely filled,**

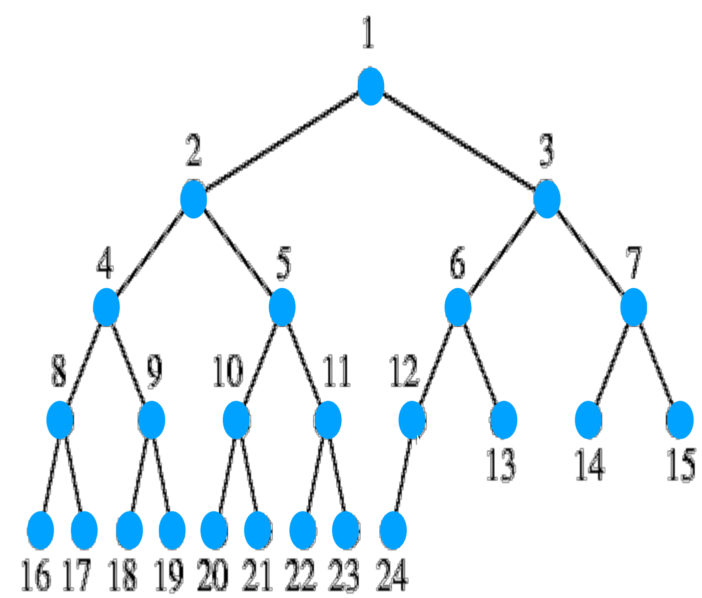
(2) **and all nodes in the last level are *as far left as possible.***

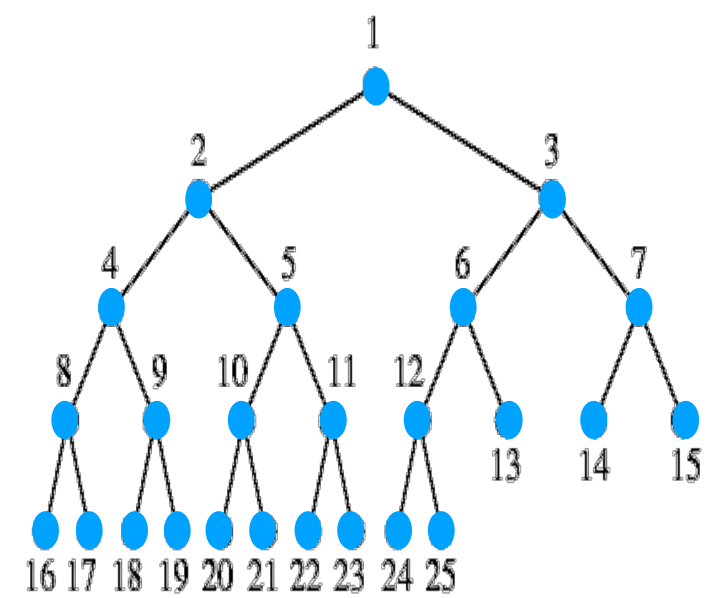
-full (proper):

every node has 0 or 2 children





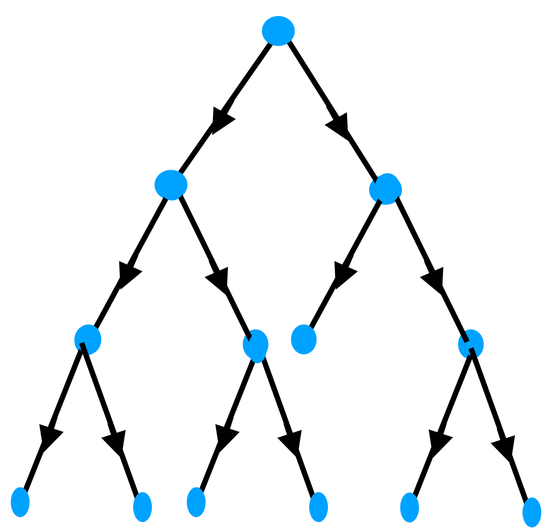


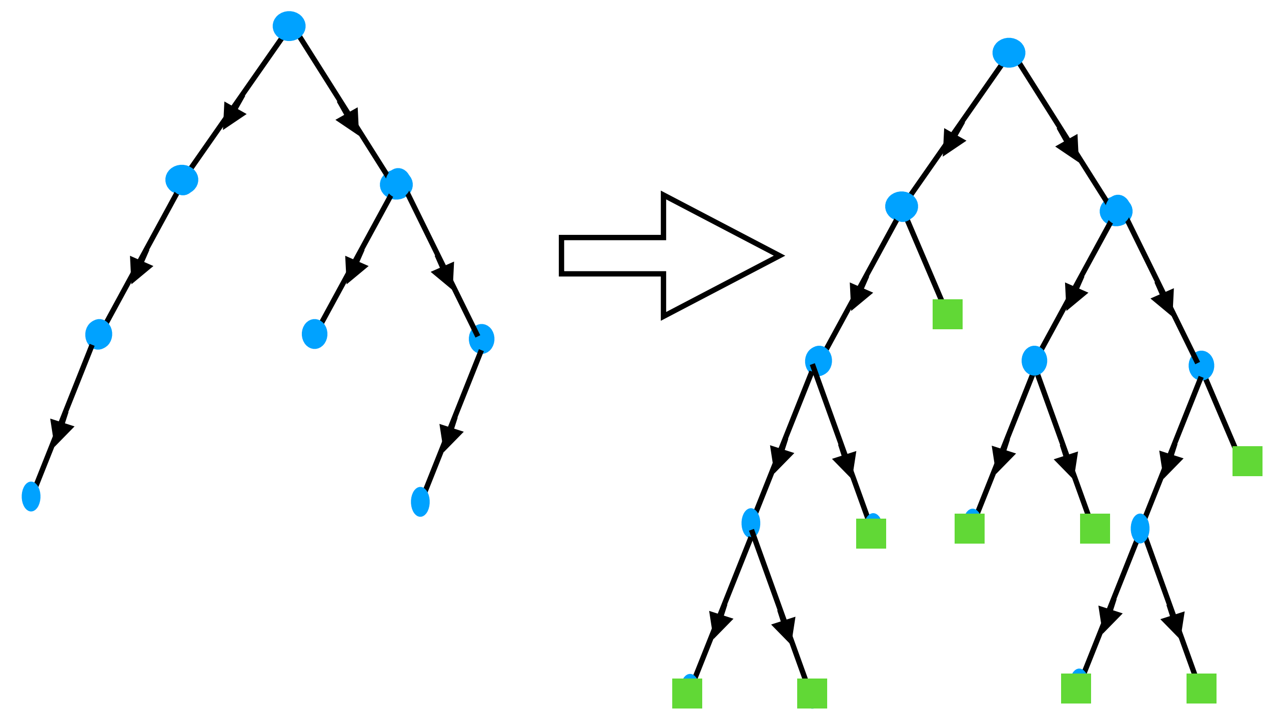


canonical…

Types of binary trees

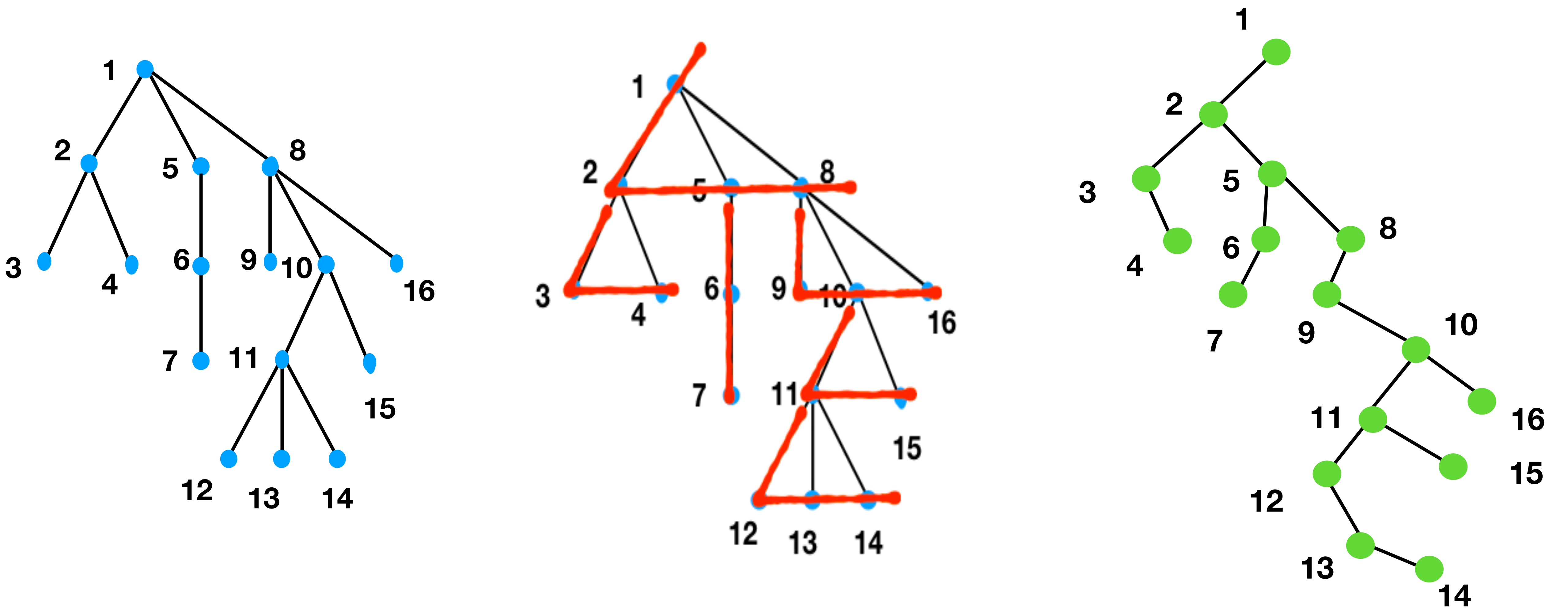
-extended binary tree (2-tree):
full tree, or extended binary tree (see Schaum)





Original notes: ***internal nodes.***
new nodes: external nodes.

First child - right sibling: encoding trees into binary trees



At each node, link children of same parent from left to right.
Parent is be linked only with the first child.

This process of converting an *k*-ary tree to an left (first) child -right sibling binary tree is sometimes called the *Knuth transform*

1-1 correspondence between ordered rooted trees and binary trees of where the root has just a a left child.

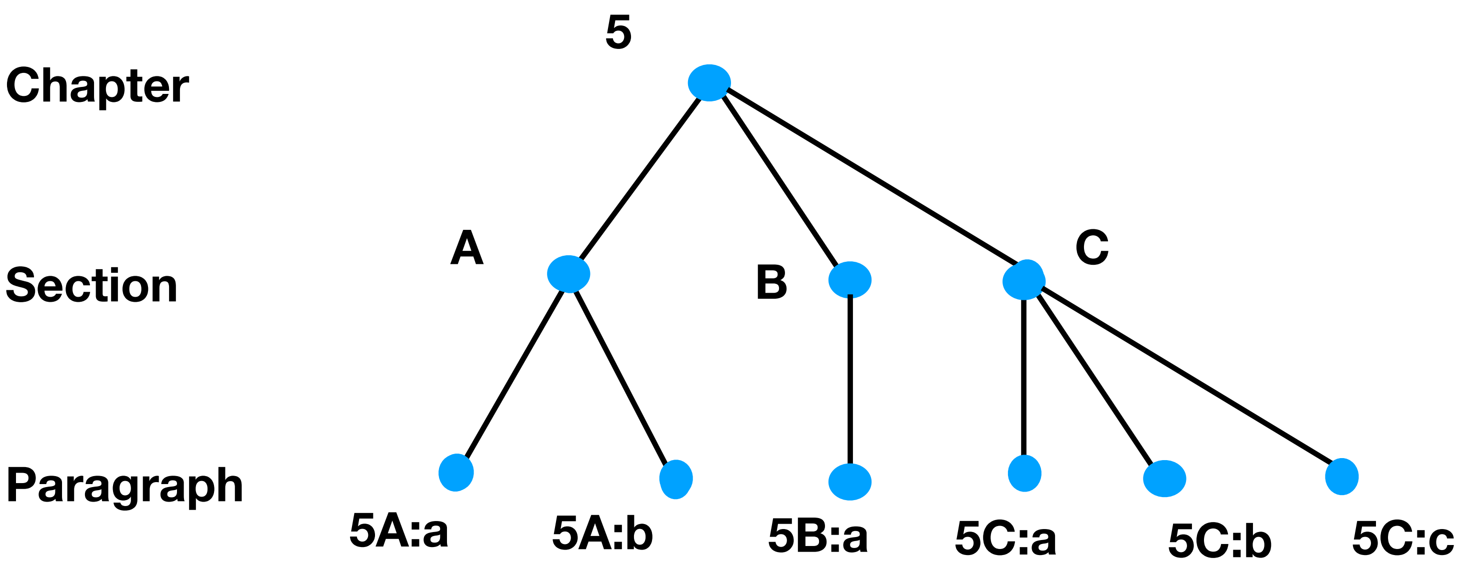
Vertex ordering and traversal in trees

Enumerating all the nodes

Example: listing out all the section headings in a book, e.g. "contents"

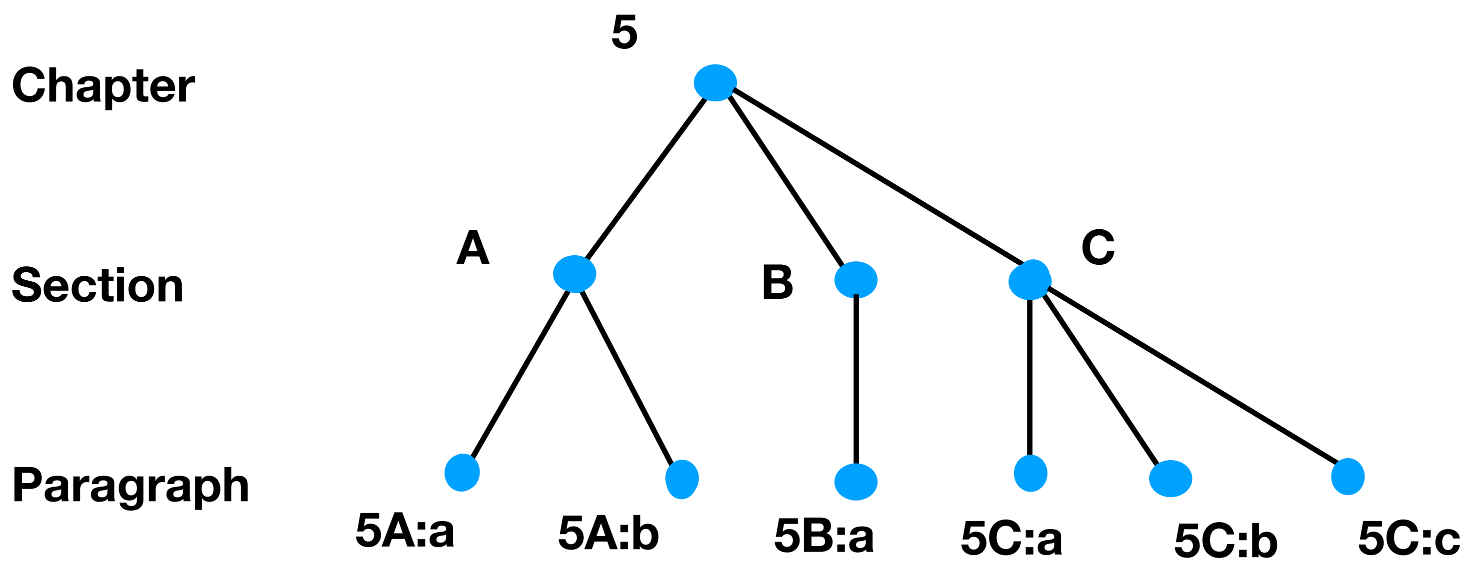
Contents:

- Chapter 5
  - -Section A
    - Para a
    - Para b
  - -Section B
    - Para a
  - -Section C
    - Para a
    - Para b
    - Para c



Enumerating all the nodes

Example: listing out all the section headings in a book, e.g. "contents"



Many other enumerations possible, e.g.:

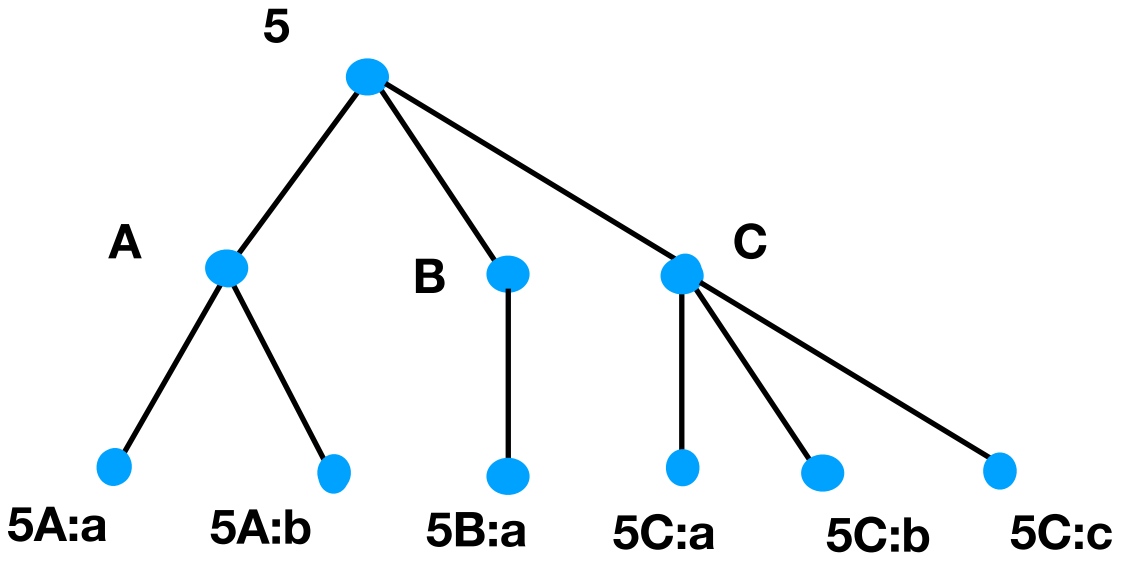
5, A, B, C, 5A:a, 5A:b, 5B:a, 5C:a, 5C:b, 5C:c

But 3 natural methods.

Preorder

"first node, then children (subtrees)"

Preorder:
(1) Process the root N.
(2) Traverse the first subtree of N in *preorder*.
(3) Traverse the second subtree of N in *preorder*.
…
(n-1) Traverse the last subtree of N in *preorder*,



5, A, 5A:a, 5A:b, B, 5B:a, C, 5C:a, 5C:b:, 5C:c

Preorder

“first node, then children (subtrees)”

Preorder:
(1) Process the root N.
(2) Traverse the first subtree of N in *preorder*.
(3) Traverse the second subtree of N in *preorder*.
…
(n-1) Traverse the last subtree of N in *preorder*,

5, A, 5A:a, 5A:b, B, 5B:a, C, 5C:a, 5C:b:, 5C:c

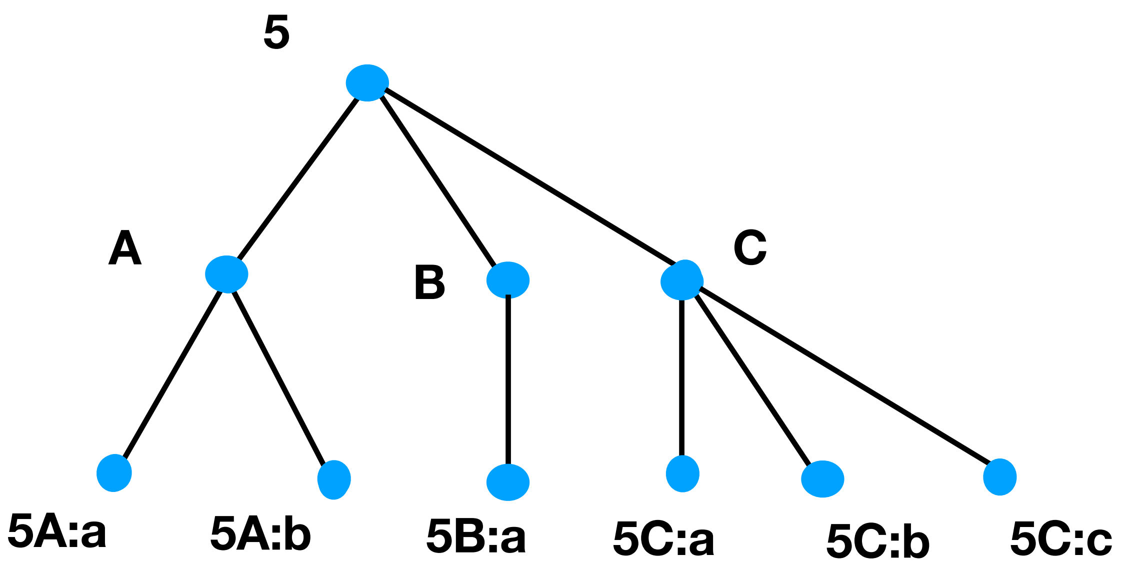
Contents:

- **Chapter 5**
  - **-Section A**
    - **Para a**
    - **Para b**
  - **-Section B**
    - **Para a**
  - **-Section C**
    - **Para a**
    - **Para b**
    - **Para c**

Preorder

"first node, then children (subtrees)"

Preorder:
(1) Process the root N.
(2) Traverse the first subtree of N in *preorder*.
(3) Traverse the second subtree of N in *preorder*.
…
(n-1) Traverse the last subtree of N in *preorder*,



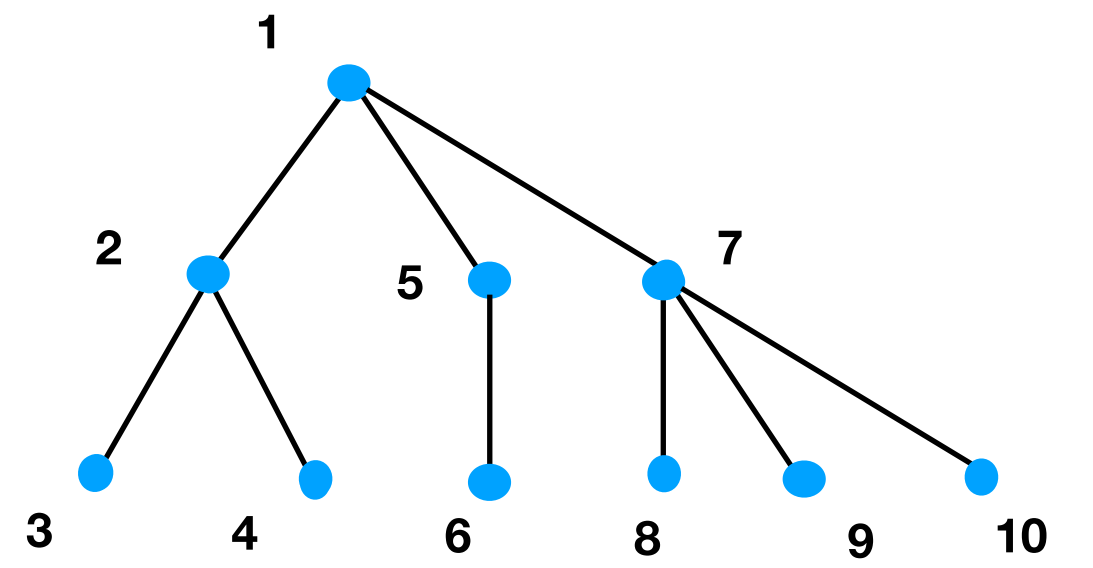
"first visit": output the vertex the first time you see it

In binary trees: NLR (node-left-right)

Preorder

“first node, then children (subtrees)”

Preorder:
(1) Process the root N.
(2) Traverse the first subtree of N in *preorder*.
(3) Traverse the second subtree of N in *preorder*.
…
(n-1) Traverse the last subtree of N in *preorder*,



Preorder enumeration

5, A, 5A:a, 5A:b, B, 5B:a, C, 5C:a, 5C:b:, 5C:c

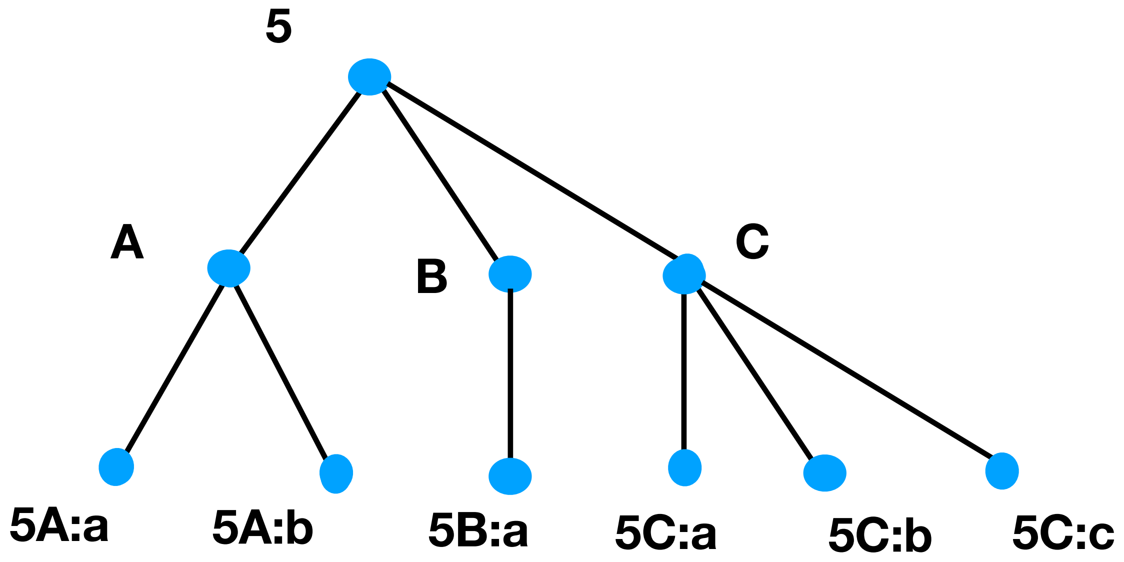
“first visit”: output the vertex the first time you see it

In binary trees: NLR (node-left-right)

Postorder

"first children (subtrees), then node"

Postorder:
(1) Traverse the first subtree of N in *postorder*.
(2) Traverse the second subtree of N in *postorder*.
…
(n) Traverse the last subtree of N in *postorder.*
(n+1) process the root N

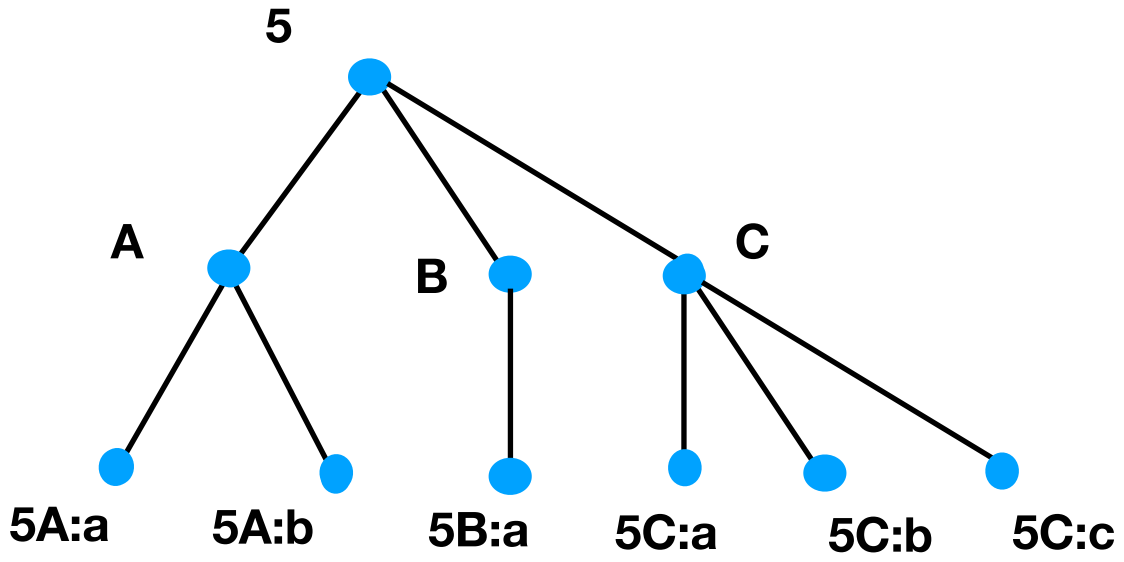


5A:a, 5A:b, A, 5B:a, B, 5C:a, 5C:b, 5C:c, C, 5

Postorder

"first children (subtrees), then node"

Postorder:
(1) Traverse the first subtree of N in *postorder*.
(2) Traverse the second subtree of N in *postorder*.
…
(n) Traverse the last subtree of N in *postorder.*
(n+1) process the root N



5A:a, 5A:b, A, 5B:a, B, 5C:a, 5C:b, 5C:c, C, 5

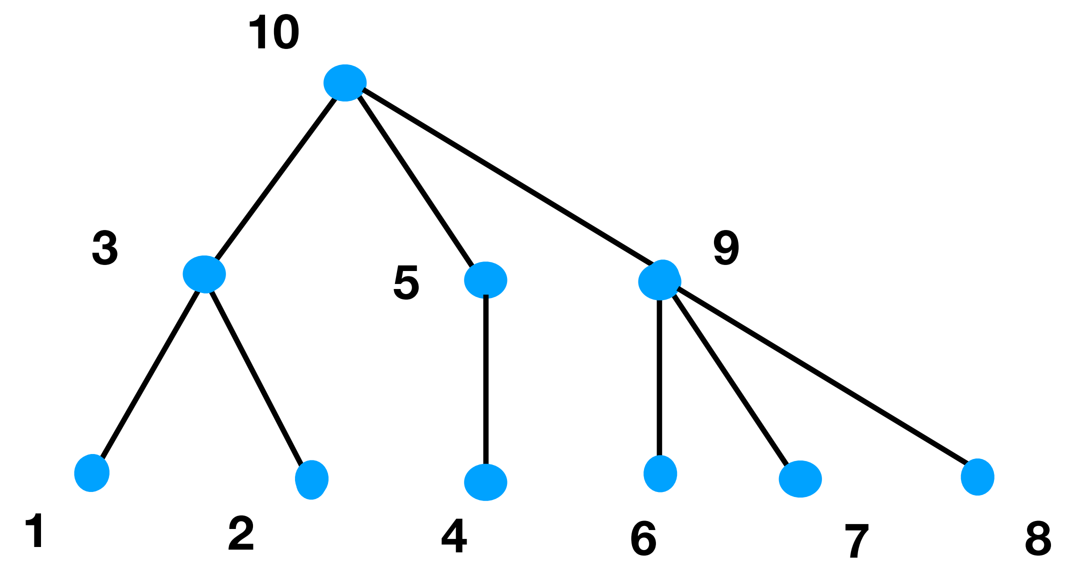
-"last visit"

-for binary trees: LRN (left-right-node)

Postorder

“first children (subtrees), then node”

Postorder:
(1) Traverse the first subtree of N in *postorder*.
(2) Traverse the second subtree of N in *postorder*.
…
(n) Traverse the last subtree of N in *postorder.*
(n+1) process the root N



Postorder enumeration

5A:a, 5A:b, A, 5B:a, B, 5C:a, 5C:b, 5C:c, C, 5

-“last visit”

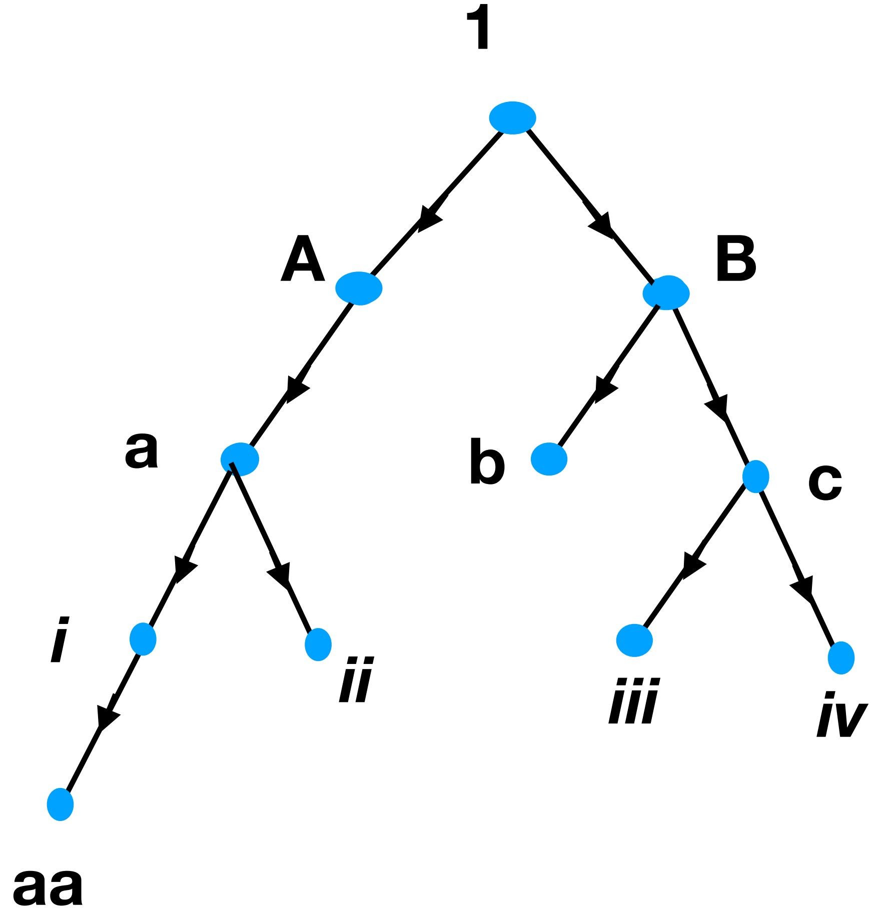
-for binary trees: LRN (left-right-node)

Inorder (symmetric ordering)

"first left child (subtree), then node, then right child (subtree)"

Works only for binary trees

Inorder:
(1) Traverse the left subtree of N in *inorder*.
(2) Traverse the root N
(3) Traverse the right subtree of N in *inorder*,

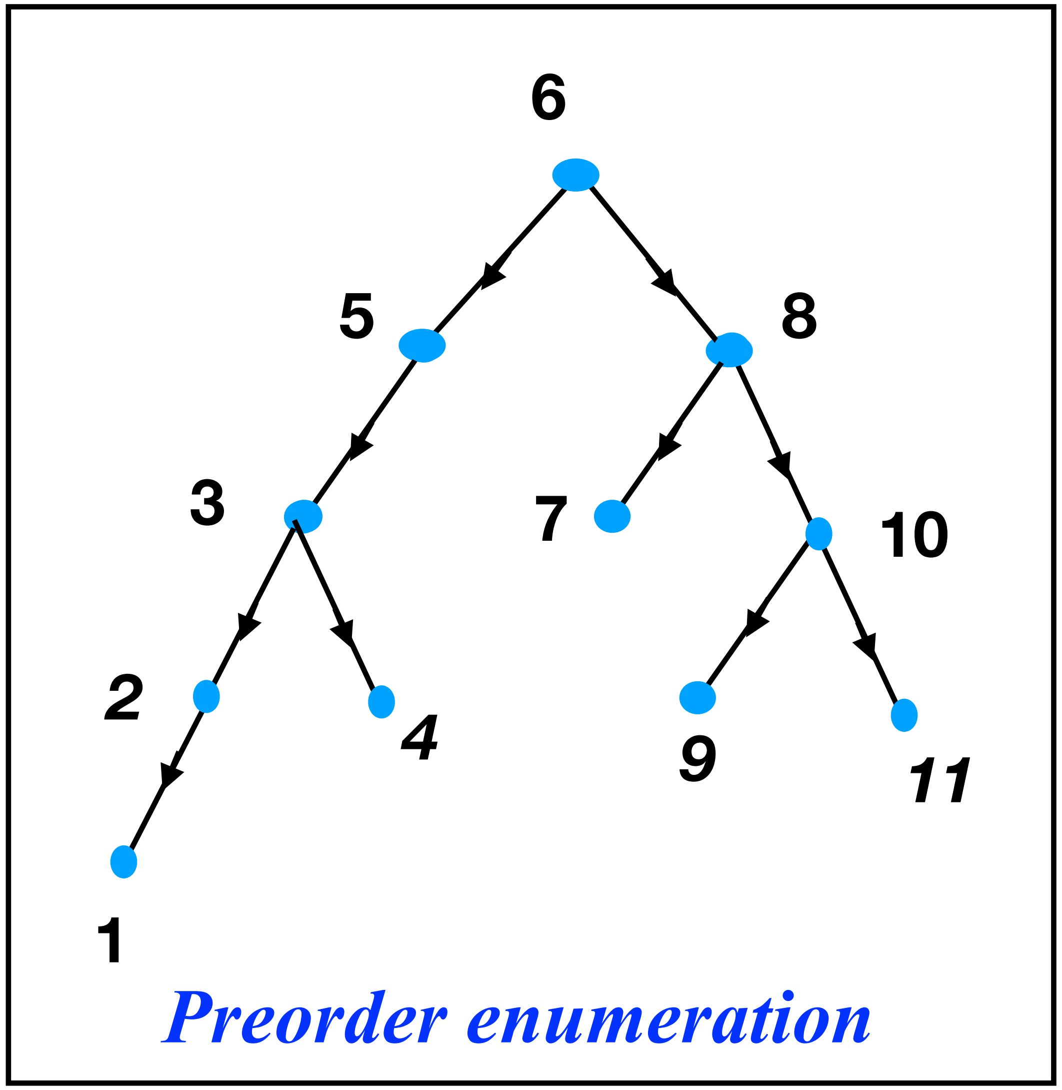


aa, i,a,ii,A,1,b,B,iii,c,iv

Inorder (symmetric ordering)

"first left child (subtree), then node, then right child (subtree)"

Works only for binary trees



Inorder:
(1) Traverse the left subtree of N in *inorder*.
(2) Traverse the root N
(3) Traverse the right subtree of N in *inorder*,

"second visit"

LNR (left-node-right)

Three main methods given recursively

preorder [NLR]: pre(T) = root(T), pre(T_1), …, pre(T_k)

postorder [LRN]: post(T) = post(T_1), …, post(T_k), root(T)

inorder [LNR]: in(T) = in(T_L), root(T), in(T_R)

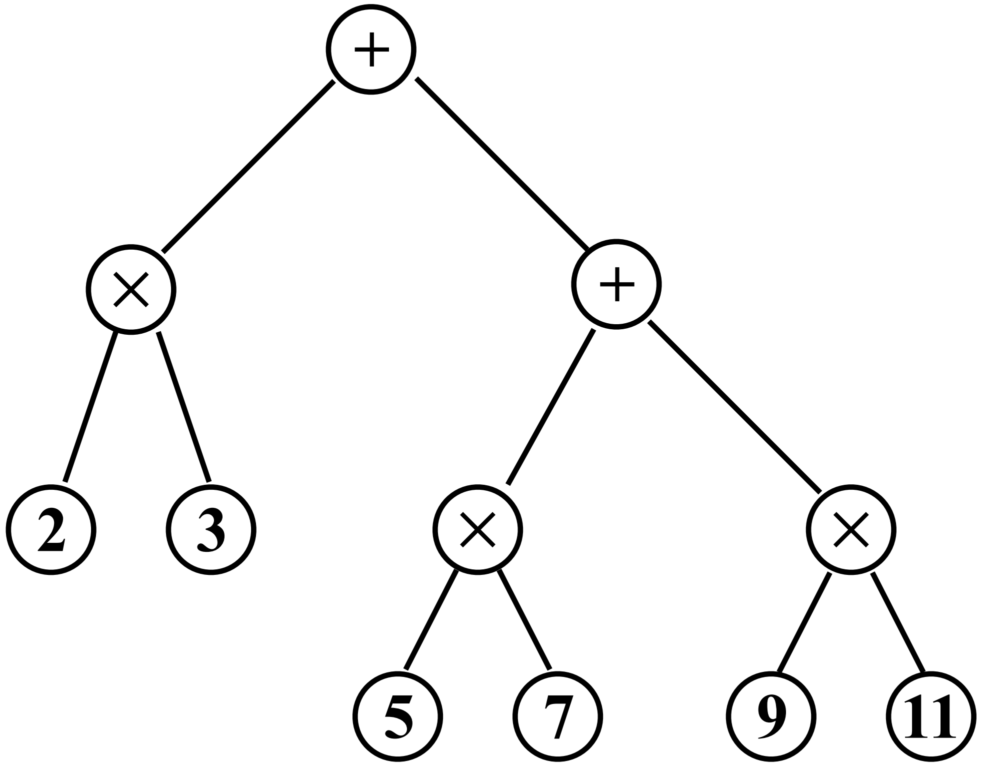
Expressions and traversal

An (algebraic) expression that only uses binary operations
can be represented by a full binary tree 2-tree (the expression tree)

Various ways to traverse the expression tree lead to 3 main ways to specify algebraic expressions

Example: from expression to tree and 3 ways back:

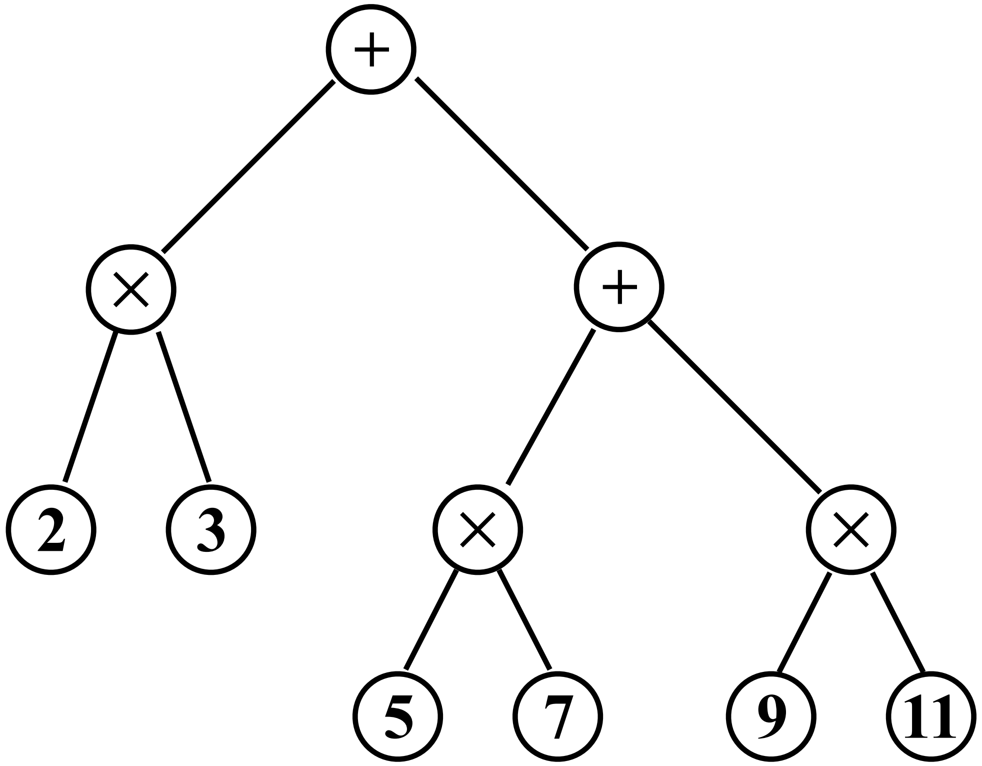
$$((2 \times 3) + ((5 \times 7) + (9 \times 11)))$$



Example: from expression to tree and 3 ways back:

$$((2 \times 3) + ((5 \times 7) + (9 \times 11)))$$

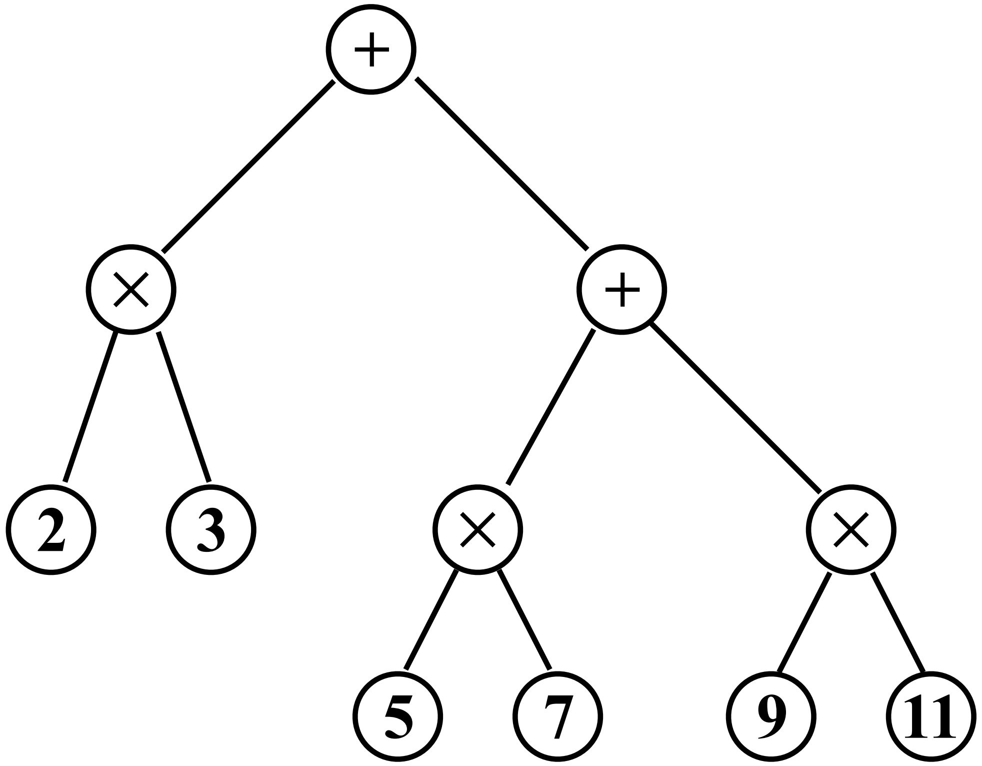
Leaves are elementary values
Inner nodes defined by brackets



Furthermore, this is *inorder traversal!*

Example: from expression to tree and 3 ways back:

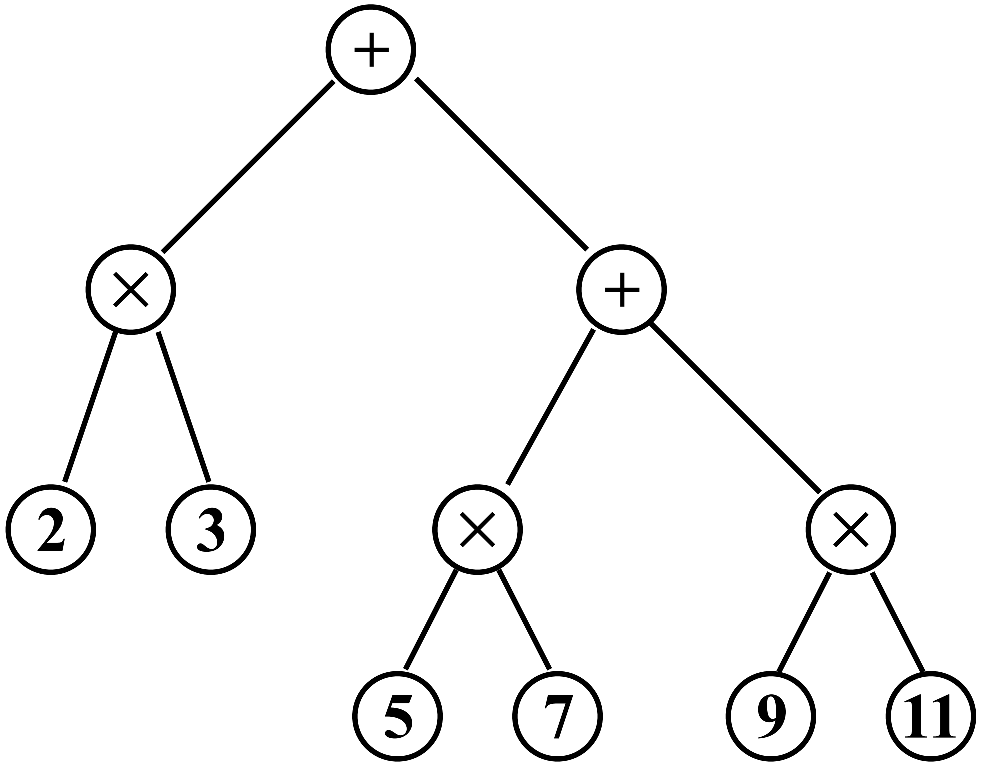
What happens if we use *Preorder (NLR)?*



Example: from expression to tree and 3 ways back:

What happens if we use *Preorder (NLR)?*

$$+ \times 2\ 3\ \ + \times 5\ 7\ \ \times\ 9\ 11$$



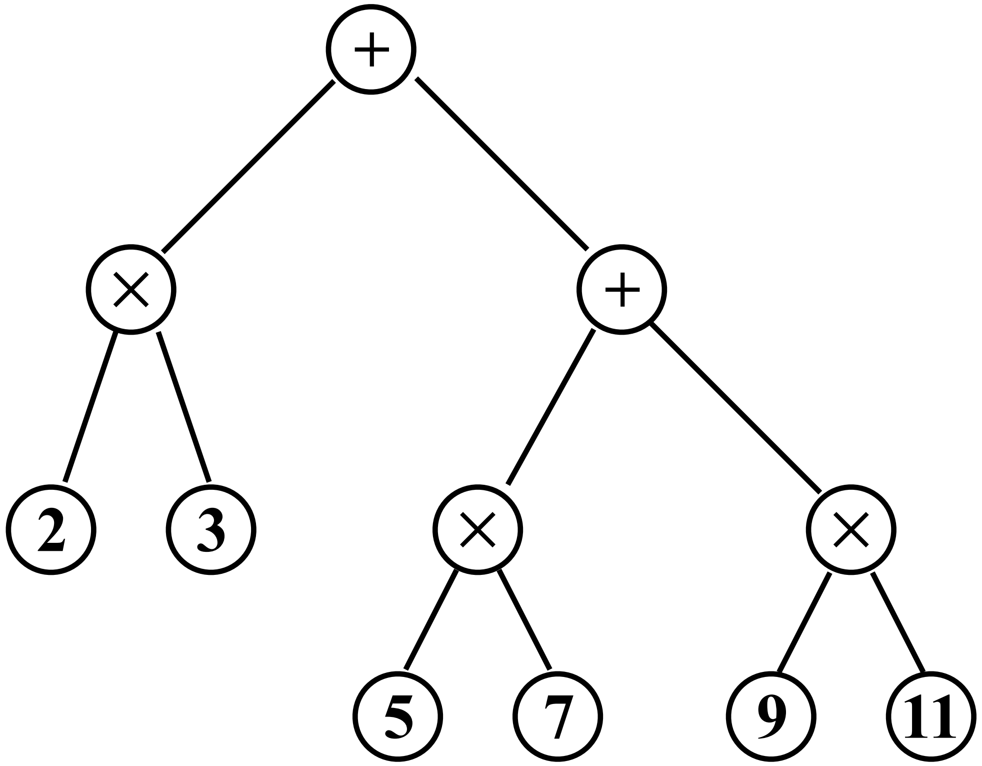
Famous *(normal) Polish notation*.
Brackets not needed.

Example: from expression to tree and 3 ways back:

Postorder (LRN)

$$2\ 3 \times 5\ 7\ \times 9\ 11\ \times + \ +$$

Famous *reverse* Polish notation.
Brackets not needed.

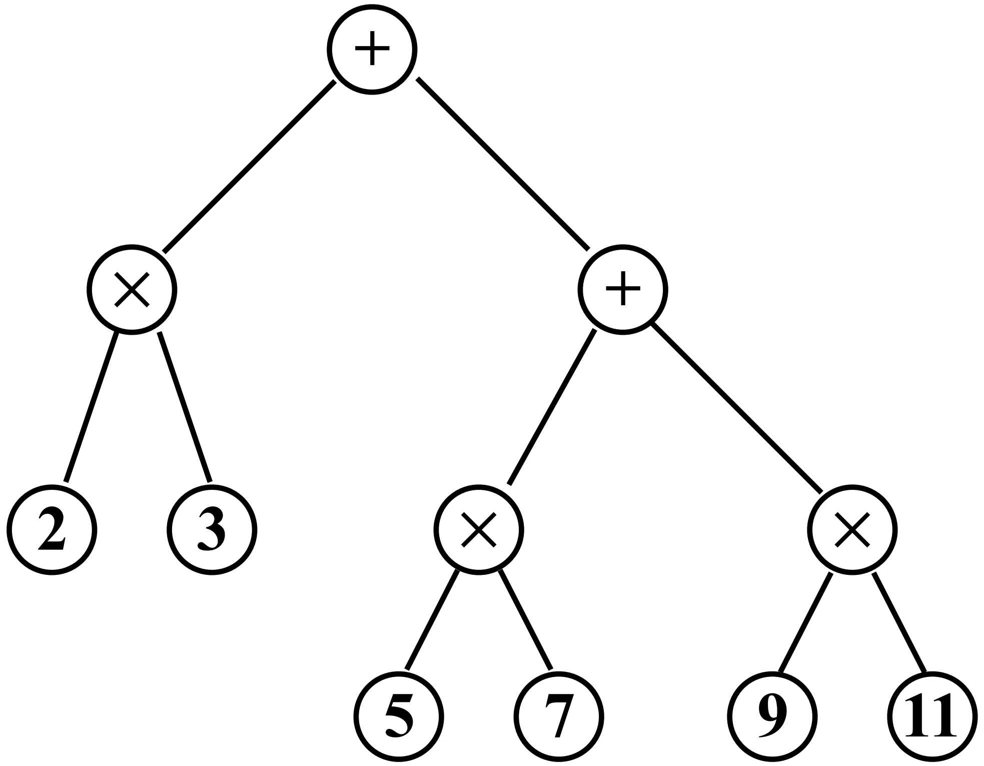


Comment:
Can work with unary operations (other arities as well), provided the arity of each operator is known.

Binary trees are a recursive structure
can provide recursive method how to evaluate expression trees

basis: for a leaf x: f(leaf) = numerical value x

inductive step: for a node @: f(node) = f(root-left-subtree) @ f(root-right-subtree)
(@ is the operation)

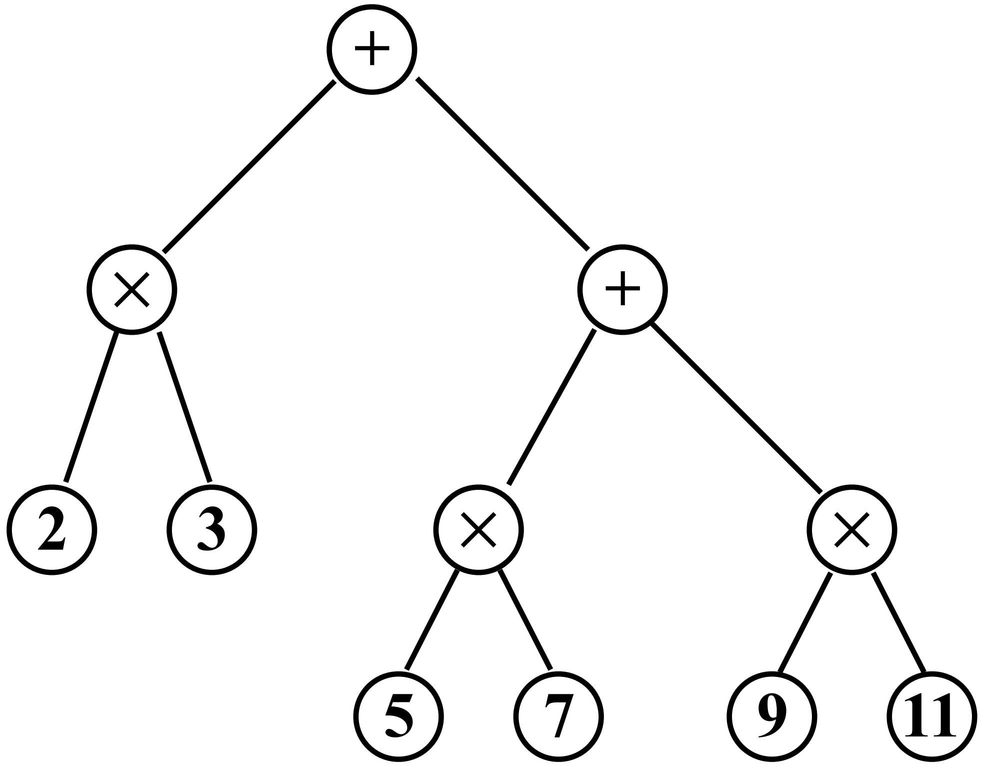


Binary trees are a recursive structure
recursively analyzing other tree properties

basis: for a leaf x: f(leaf) = 1

inductive step: for a node y: f(node) = f(root-left-subtree) + f(root-right -subtree) + 1

what does this compute ?

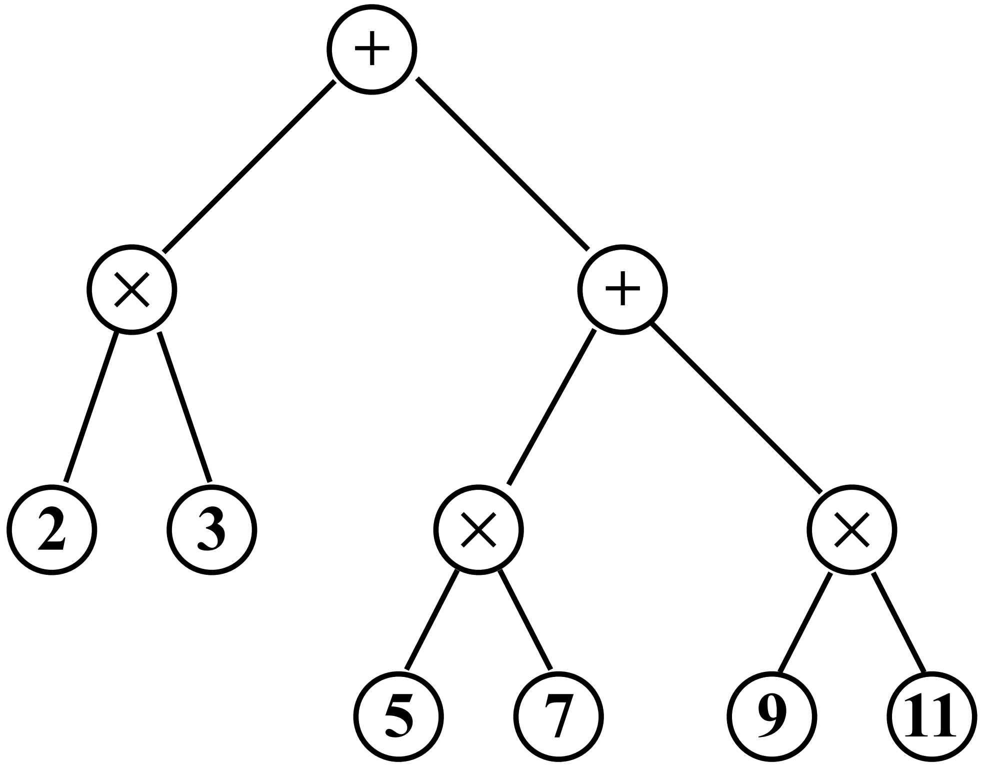


Binary trees are a recursive structure
recursively analyzing tree properties

basis: for a leaf x: f(leaf) = 1

inductive step: for a node y: f(node) = f(root-left-subtree) + f(root-right-subtree) + 1

=tree size



Modulo computation and equivalence relations

Schaum: 2.8, 11.5, 11.8

Equivalence relation: a binary relation which is simultanously

(1) reflexive (xRx, *for all* x)

(2) symmetric (xRy *implies* yRx)

(3) transitive (xRy and yRz imply xRz)

Capture equalities up to (disregarding) some properties:

-“same colour”

-=

-parity

-“being parallel" for lines

Modulo computation examples

- 12h clock.
  It is 5 o'clock; What time is it 8 hours later?

- 24h clock.
  It is 22.30; What time is it 3 h later?

- 24h clock & 60 minute hour.
  It is 22.30; What time is it 2h 33mins later?

- 7 day weeks:
  Mon = 1st day; Tue = 2nd day; … Sun = 7th day.
  What day comes 20 days after a Tuesday?

Its about remainders when dividing with integers!

Congruence modulo *n*

Def. For integers *a, b, n*, n>0, we say *a is congruent with b modulo n*, written

$$a \equiv b \ (mod\ n)$$

if *n* divides the difference *(a-b)* (in other words, *n* | *(a-b)*).
***n* is called the modulus.**

NB:

-*a* is congruent with *b* mod *n if and only if a and b have the same remainder when divided with n. Eg: how many minutes past full hour. (245 = 305)*

-Specially, if b is the remainder if a div n.

For any fixed n, "congruence mod n" is a relation (*a* and *b* are in that relation).

It is an equivalence relation: (1) reflexive, (2) symmetric, (3) transitive.

Congruence modulo n

***Congruence mod n* is an equivalence relation.**

Proof:

(1) reflexive $a \equiv a$ because $n \mid 0$ always.

(2) symmetric $a \equiv b \Rightarrow n \mid (a-b) \Leftrightarrow a-b = k \cdot n$ $(k \in \mathbb{Z})$

$\Leftrightarrow b-a = (-k) \cdot n \Leftrightarrow b \equiv a \mod n$ ✓

(3) $a \equiv b \pmod n$ & $b \equiv c \pmod n$

$\Rightarrow n \mid (a-b)$, $n \mid (b-c) \Rightarrow n \mid (a-b)+(b-c) \Rightarrow n \mid a-c$

$\Rightarrow a \equiv c \pmod n$. ✓

As we said:

Equivalence relations capture equalities “up to" some details.

“up to *X*" can be taken to mean "disregarding a possible difference in *X*"

(graph isomorphism is equivalence “up to” permutation of labels,
equivalence relation “strings of equal lenght” ignores all except lenght…)

Equivalence class groups together all the elements that “are the same” according to the given equivalence relation.

Such a set is specified by one representative element.

Equivalence classes.

Let R be an equivalence relation on V, let x be some element of V.

With $[x]_R = \{y \in V \mid xRy\}$ *we denote the equivalence class of* x *with respect to the relation* R

Equivalence class groups together all the elements that “are the same” according to the given equivalence relation.

Such a set is specified by one representative element.

Residue classes mod n are equivalence classes relative to the relation of *congruence mod n*

Consider the equivalence relation R = *congruence* ***mod 7***

$$[0]_R = \{\ldots -14, -7, \quad 0, 7, 14, 21, \ldots\}$$

$$[1]_R = \{\ldots -13, -6, \quad 1, 8, 15, 22, \ldots\}$$

$$[2]_R = \{\ldots -12, -5, \quad 2, 9, 16, 23, \ldots\}$$

$$[3]_R = \{\ldots -11, -4, \quad 3, 10, 17, 24, \ldots\}$$

$$\vdots$$

$$[6]_R = \{\ldots -8, -1, \quad 6, 13, 20, 27, \ldots\}$$

$$[7]_R = ?$$

[x]..
take x see what
remainder when div 7
collect all numbers with same
remainder

Equivalence class [x] is also denoted

Residue classes mod n are equivalence classes relative to the relation of *congruence mod n*

Residue class [x] is also denoted $\bar{x}$

$$[0]_R = \{\ldots -14, -7, \quad 0,7,14,21,\ldots\} = \bar{0}$$

$$[1]_R = \{\ldots -13, -6, \quad 1,8,15,22,\ldots\} = \bar{1}$$

$$[2]_R = \{\ldots -12, -5, \quad 2,9,16,23,\ldots\} = \bar{2}$$

$$[3]_R = \{\ldots -11, -4, \quad 3,10,17,24,\ldots\} = \bar{3}$$

$$\vdots$$

$$[6]_R = \{\ldots -8, -1, \quad 6,13,20,27,\ldots\} = \bar{6}$$

[x]..
take x see what
remainder when div 7
collect all numbers with same
remainder

Mini example

Consider the equivalence relation R = *congruence* ***mod 2***

How many residue classes?

If I give you a number in binary, how can you tell what class it is in?

What about mod 4?
If I give you a number in binary, how can you tell what class it is in?

Modulo arithmetic

Theorem. Suppose that $a \equiv b \ (mod\ n)$ **and** $c \equiv d \ (mod\ n)$**. Then**

(1) $a + c \equiv b + d \ (mod\ n)$
(2) $a - c \equiv b - d \ (mod\ n)$
(3) $a \times c \equiv b \times d \ (mod\ n)$

Corollary. If $a \equiv b \ (mod\ n)$ **then** $a^k \equiv b^k \ (mod\ n)$ **for all integer** $k>0$**.**

Example of use: last digit

What is the last digit of 3^{234}?

Note: last digit of x the remainder of x divided by 10.

What $0 \leq b < 10$ is 3^{234} congruent mod 10?

Use $e.g.\ 3^4 = 81,\ so\ 81 \equiv 1\ (mod\ 10)$. (dropping mod 10 notation…)

$3^{234} = 3^{4\times58+2}$ $\quad$ $3^{4\times58} = (3^4)^{58} \equiv 1^{58} \equiv 1$

$3^2 = 9 \equiv 9$ $\quad$ $3^{234} = 3^{4\times58+2} = (3^{4\times58} \times 3^2) \equiv 1 \times 9 = 9$

Example of use: days of the week

1-Jan-2000 - Sat

2-Jan-2000 - Sun

31-Feb-2000 - …

31-Dec-2000 - Sun

1-Jan-2001 - Mon

13-May-2023 - ?

Find out how many days ahead of a date with known day

1	1-Jan-2000 - Sat
2	2-Jan-2000 - Sun
31	31-Jan-2000 - …
32	1-Feb-2000 - …
365	31-Dec-2000 - Sun
	1-Jan-2001 - Mon
x	13-May-2023 - ?

x *mod* 7 gives you the solution…

Compute number of days…

1	1-Jan-2000 - Sat
2	2-Jan-2000 - Sun
31	31-Jan-2000 - …
32	1-Feb-2000 - …
365	31-Dec-2000 - Sun
	1-Jan-2001 - Mon
x	13-May-2023 - ?

23 full years = 23 x 365
6 leap years = +6
Jan - May = 31+28+31+30
13 = +13

= 8534 (mod 7)

Compute number of days…

1	1-Jan-2000 - Sat
2	2-Jan-2000 - Sun
31	31-Jan-2000 - …
32	1-Feb-2000 - …
365	31-Dec-2000 - Sun
	1-Jan-2001 - Mon
x	13-May-2023 - ?

23 full years = 23 x 365
6 leap years = +6
Jan - May = 31+28+31+30
13 = +13

= 8534 (mod 7)

but can compute it in parts!

=2 x 1 + 6 + 3+0+3+2+6=22=1

Started Sat + 1 = Sun!

Theorem. Suppose that $a \equiv b \ (mod\ n)$ and $c \equiv d \ (mod\ n)$. Then

(1) $a + c \equiv b + d \ (mod\ n)$
(2) $a - c \equiv b - d \ (mod\ n)$
(3) $a \times c \equiv b \times d \ (mod\ n)$

Corollary. If $a \equiv b \ (mod\ n)$ then $a^k \equiv b^k \ (mod\ n)$ for all integer *k>0*.

100^{102} **mod** 13

$$100^{102} \equiv 9^{102} \equiv (81)^{51} \equiv (3)^{51} \equiv (27)^{17} \equiv 1^{17} \equiv 1$$

41^{2016} **mod** 13

$$41^{2016} \equiv 2^{2016} \equiv 16^{504} \equiv 3^{504} = 27^{167} = 1$$

$$100^{102} + 41^{2016} \equiv 1 + 1 = 2$$

A trick you may use (not necessary)

Little Fermat's theorem

Theorem. For any prime p *and any integer a:*

$a^{p-1} \equiv 1 \ (mod\ p).$

Eg. 100^102 mod 13 = 100^(8*12 + 6) = [100^(12)]^8 * 100^6 = 100^6 = 10^12 = 1

Note, the questions of divisibility
("is x divisible by y") is the question is:
$x \equiv 0 \ (mod \ \ y)$

Partitions and equivalence classes

Def. Given a set V, the set $\{V_1, \dots V_k\}$ of subsets of V is called a partition of V if

(1) (pariwise disjointness) $V_i \cap V_j = \emptyset, \text{ for all } i \neq j$

(2) (cover) $\bigcup_{i=1}^{k} V_k = V$

in other words, every x from V is in exactly one subset V_j

Residue classes partition the set of integers

Theorem. Let $[0]_R, \ldots [k-1]_R,$ be the residure classes with respect to the equivalence relation congruent modulo k.
Then $\{[0]_R, \ldots [k-1]_R\}$ is a partition of $\mathbb{Z}$.

NB, partitions can be infinite:

$V_k = \{l \times 2^k \mid l \text{ is odd}\}, \quad k \geq 0$

Computing using residual classes

Consider the equivalence relation R = *congruence* ***mod n***

$$[x]_R = \{y \mid xRy\} = \{an + k \mid a \in \mathbb{Z}, k \text{ } is \text{ } remainder \text{ } of \text{ } dividing \text{ } x \text{ } with \text{ } n\}$$

$y \in [x]_R$, **such that** $y \neq x$

nonetheless:

$$[x]_R = [y]_R$$

Representative does not matter: *well-defined*

Computing using residual classes

Can do arithmetic!

$$[x]_R + [y]_R = [x + y]_R$$

E.g. mod 7. [3] + [3] = [6]; [3] + [4] = [7] = [0].

Mathematically, this is addition mod 7 (n).

$$\textit{Also notation}: \bar{x} + \bar{y} = \overline{x + y}$$

Computing using residual classes

Can do arithmetic!

$$[x]_R \times [y]_R = [x \times y]_R$$

E.g. mod 7. [3] x [3] = [9] = [2]; [3]x[4] = [12] = [5].

Mathematically, this is multiplication mod 7 (*n*).

$$Also\ notation: \bar{x} \times \bar{y} = \overline{x \times y}$$

Computing using residual classes

Interesting thing happens when *n* is not prime…

$[x]_R \times [y]_R = [x \times y]_R$

E.g. *mod 6.*
[3] x [2] = [6] = [0]

So modular arithmetic behaves almost the same but,

if the *order (n) is not prime* $a \times b \ (mod\ n) \not\Rightarrow a = 0 \ or \ b = 0.$

if *n is prime all good.*

Computing using residue classes

We denote these structures (of mod n) arithmetic $\mathbb{Z}_n$

On one hand, elements are integers, and operations are mod n

On the other, the elements of $\mathbb{Z}_n$ are residue classes (subsets) [k].

These structures are isomorphic.

Computing using residue classes

$\mathbb{Z}_6$

+	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

·	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

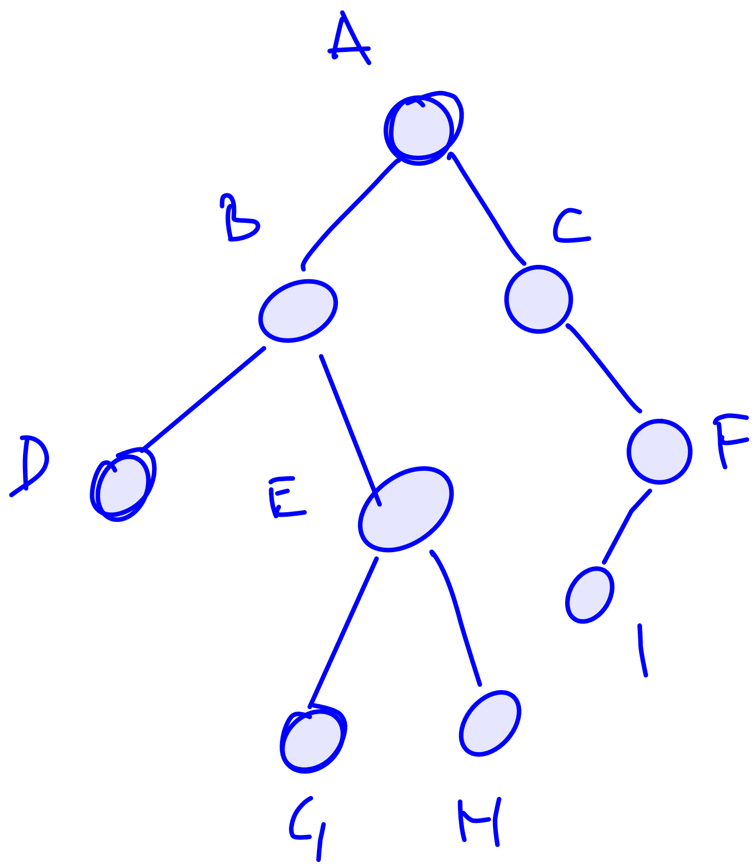
Computing using residue classes

$\mathbb{Z}_7$

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

·	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Exercises:

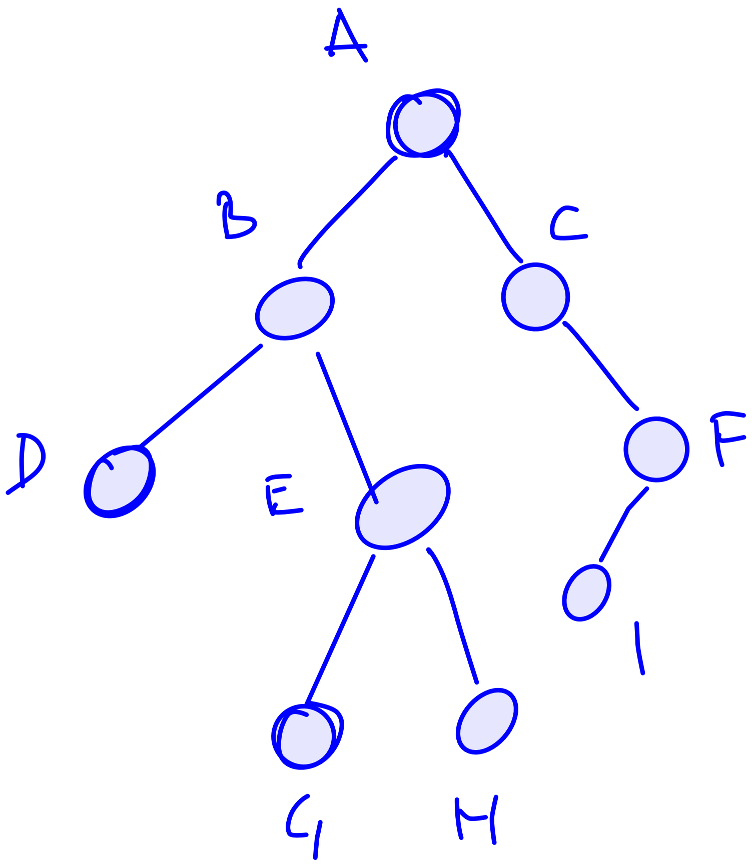


(i) NLR (PREORDER)

(ii) LNR (INORDER)

(iii) LRN (POSTORDER)

Exercises:



(i) NLR (PREORDER)

(ii) LNR (INORDER)

(iii) LRN (POSTORDER)

A B D E G H C F I

D B G E H A C I F

D G H E B I F C A

$x^2 - 1$ is divisible by 8, for odd x

inductive

basis 0

$(x+2)^2 - 1$

$= x^2 + 4x + 4 - 1$

$= \underbrace{x^2 - 1} + 4x + 4$

$= \underbrace{x^2 - 1}_{\checkmark} + \underbrace{4 \underbrace{(x+1)}_{\text{even}}}_{\checkmark}$

direct

$= (x-1)(x+1)$

$\uparrow$ even $\quad \uparrow$ even

and one is div with 4.

$x^2 - 1 \equiv 0 \mod 8$

$x^2 \equiv 1 \mod 8$

$[1] \to 1^2 = 1 \equiv 1 \mod 8$

$[3] \quad 3^2 = 9 \equiv 1 \mod 8$

$[5] \quad 5^2 = 25 \equiv 1$

$[7] \quad 7^2 = 49 \equiv 1$

elements $5 + 8k$

$\Rightarrow (5+8k)^2 = \underbrace{(8k)^2}_{\checkmark} + \underbrace{5 \times 8k}_{\checkmark} + \underbrace{25}$ $\checkmark$

Is

$224 \equiv \underline{768} \mod 8$

In $\mathbb{Z}_{13}$. with $-a$ we denote $x \in \mathbb{Z}$

st $x + a \equiv 0 \pmod{13}$

-3?

In $\mathbb{Z}_{17}$ -15 ?