University of Michigan-Dearborn

Chapter 9 : Trees

UNIVERSITY OF MICHIGAN-DEARBORN

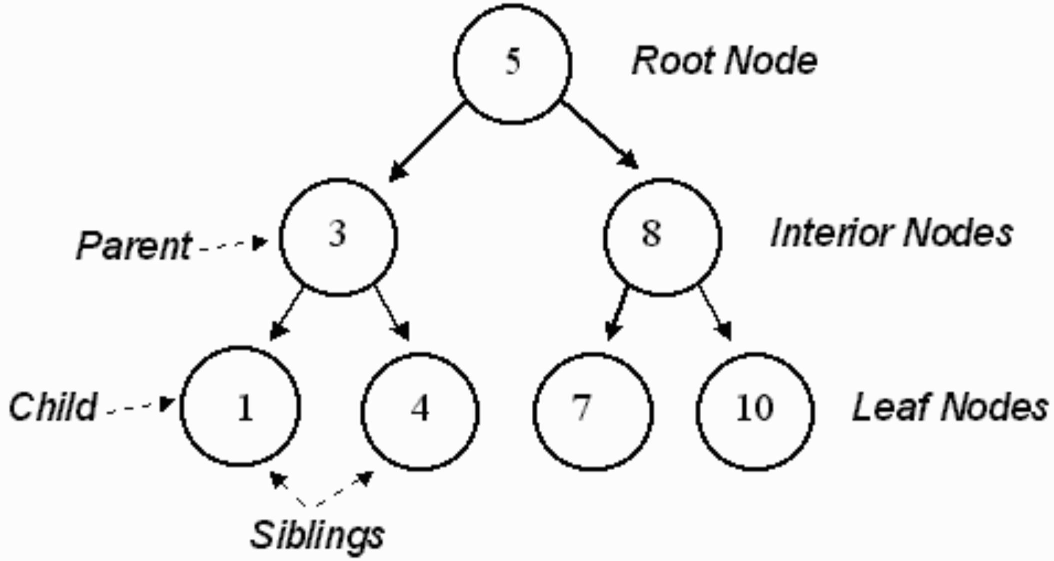
Introduction

Trees

- **Definition:** A tree is a simple graph in which there is a unique path between every pair of vertices.
- Computer Science, in particular, makes extensive use of trees.
- In Computer Science, Trees are useful in organizing and relating data in a database.

Trees

- ***Example of Tree:***

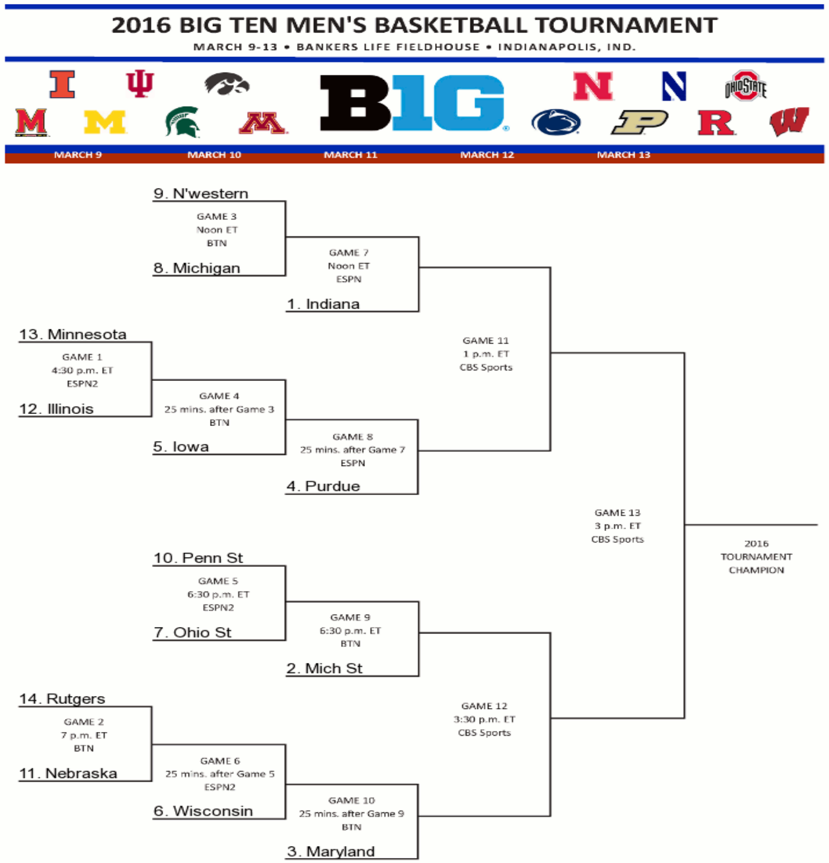


A tree is an abstract data type:

- One entry point, The **root**.
- Each node is either a **leaf** or an internal node.
- An internal Node has one or more **children**, nodes that can be reached directly from that internal node.
- The internal node is said to be the **parent** of its child nodes.

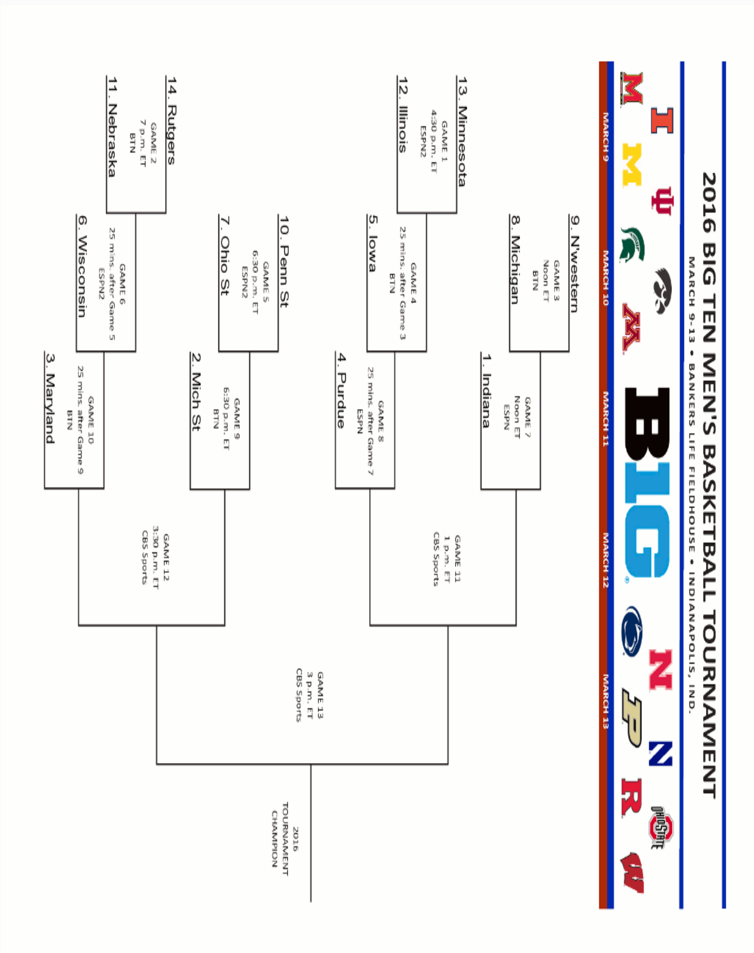
Trees

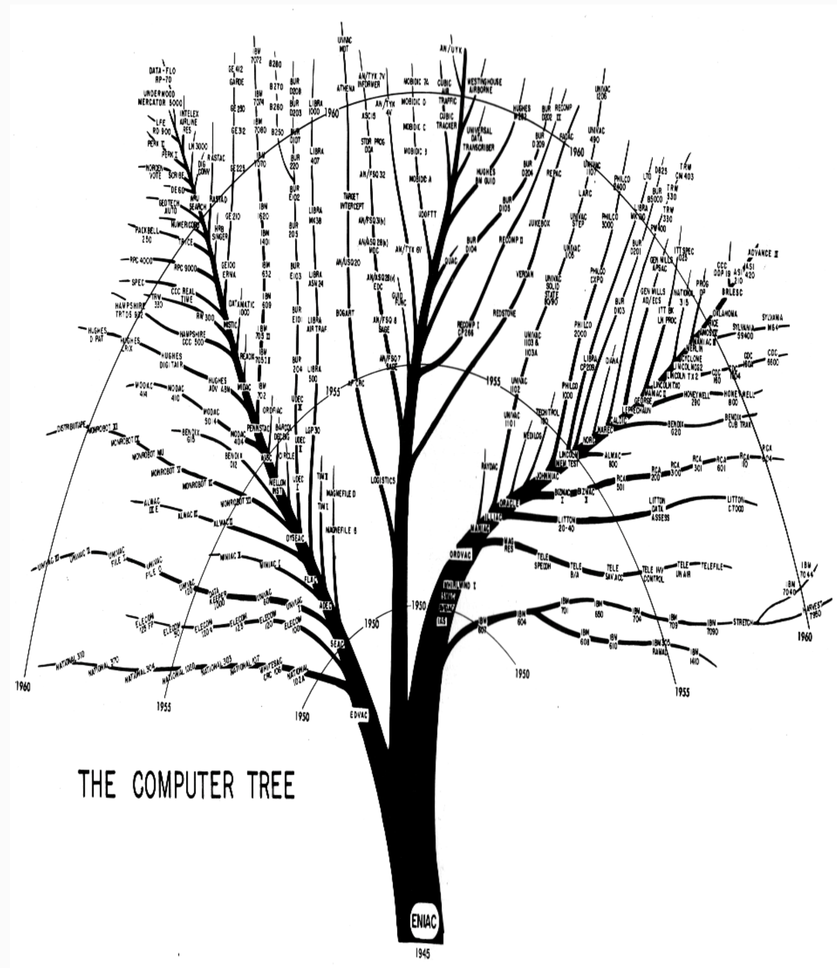
- **Example:** A common form of tree used in everyday life is the tournament tree, used to describe the outcome of a series of games, such as Basketball.



Trees

- If we rotate the previous figure, it looks like a natural tree.





Trees

- **Definition:**

✧ A (free) *tree T is a simple graph* satisfying the following:

1) v and w are vertices in T
2) there is a unique simple path from v to w

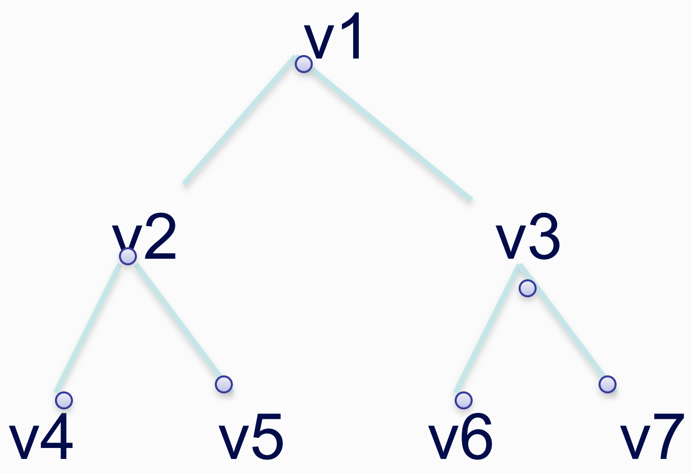
✧ A *rooted tree* is a tree in which a particular vertex is designated the root.

✧ The *level of a vertex* v is the length of the simple path from the root to v.

✧ The *height* of a rooted tree is the maximum level number that occurs.

Trees

- **Example:**
  - ❖The vertices v1, v2, v3, v4, v5, v6, v7 in the rooted tree of this figure are on (respectively) levels 0, 1, 1, 2, 2, 2, 2.
  - ❖The height of the tree is 2



Trees

- Example: Huffman Codes

✓ The most common way to represent characters internally in a computer is by using fixed-length bit strings.

✓ For Example: ASCII (American Standard Code for Information Interchange) represents each character by a string of seven bits.

✓ Example of ASC II code :

A ,, Z = $(41)_{16}$, …, $(5A)_{16}$

a,,,z = $(61)_{16}$,.., $(7A)_{16}$

'0' ,,,'9' = $(30)_{16}$,,, $(39)_{16}$

Trees

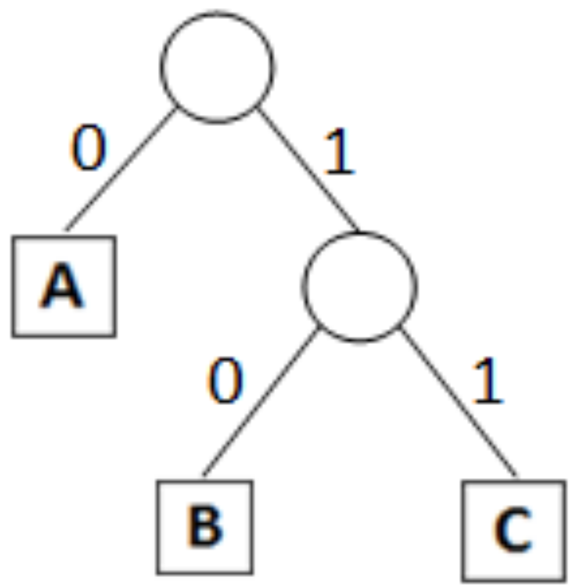
The ASCII table:

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Trees

Huffman code represents characters by variable-length bit strings. The idea is to use short bit strings to represent the most frequently used characters and to use longer bit strings to represent less frequently used characters.



A: 0

B: 10

C: 11

Trees

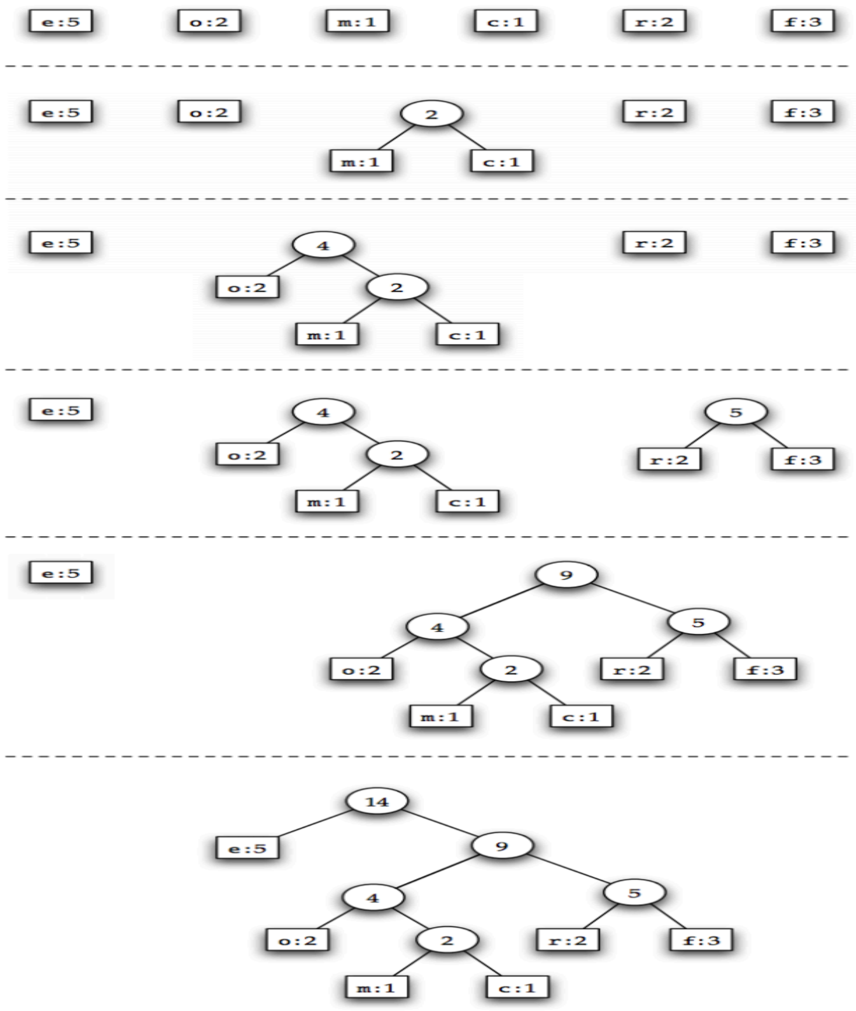
The first step of *Huffman encoding* is building the Huffman tree. Given a set of characters and their associated frequencies, we can build an optimal Huffman tree as follows:

- Construct leaf Huffman trees for each character/frequency pair
- Repeatedly choose two minimum-frequency Huffman trees and join them together into a new Huffman tree whose frequency is the sum of their frequencies.
- When only one Huffman tree remains, it represents an optimal encoding.

Then the code for each character can be obtained by following the path from the root of the tree to the leaf holding the given character, assigning and accumulating a '0' when following a left edge and a '1' when following a right edge. The accumulated zeros and ones at each leaf constitute a Huffman encoding for those symbols. The image below illustrates this process.

Tress

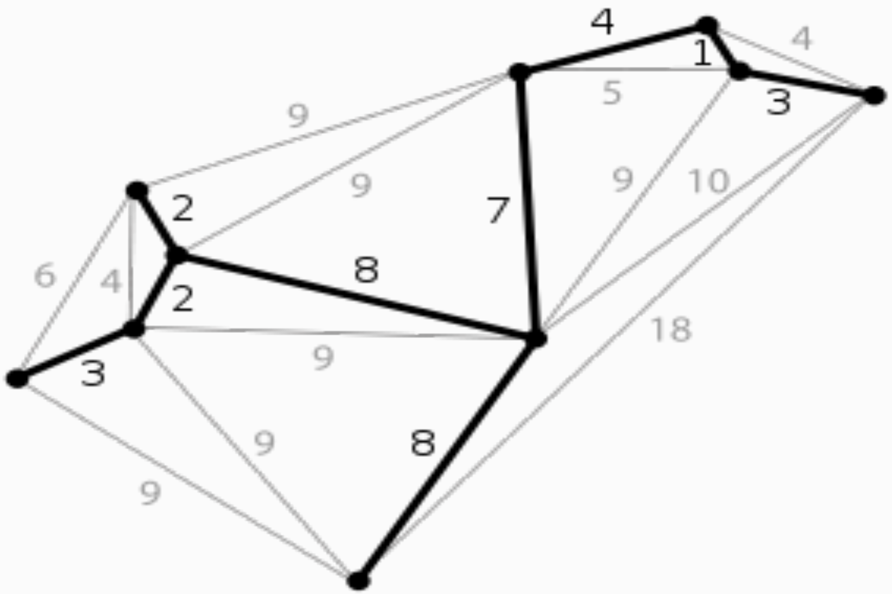
- Example: Construction an optimal Huffman Code.



Spanning Trees

Trees

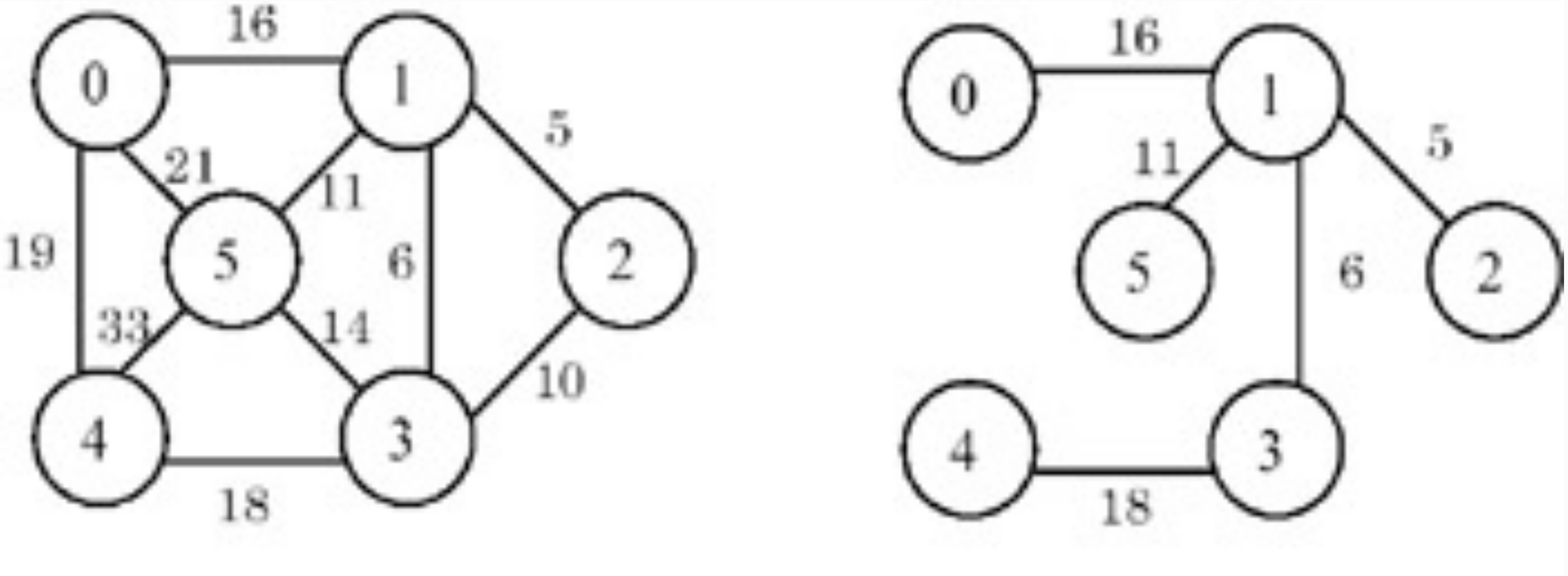
- **Definition:** A tree T is a *spanning tree* of a graph G if T is a subgraph of G that contains all of the vertices of G
- **Example:**



A planar graph and its minimum spanning tree. Each edge is labeled with its weight, which here is roughly proportional to its length.

Trees

- **Definition:** A **minimum spanning tree** (**MST**) or **minimum weight spanning tree** is then a spanning tree with weight less than or equal to the weight of every other spanning tree.
- **Example 1:**

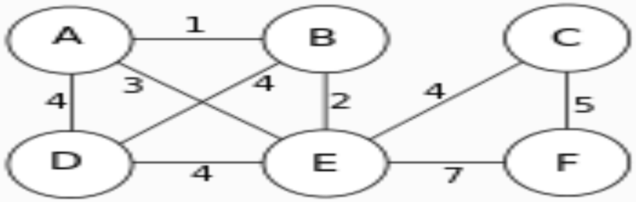


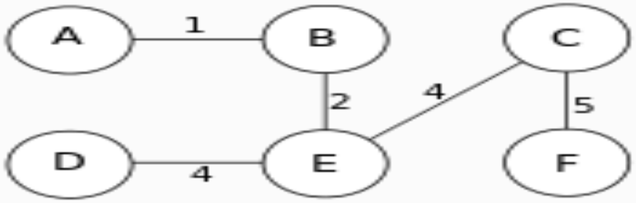
An example graph and its minimum spanning tree.

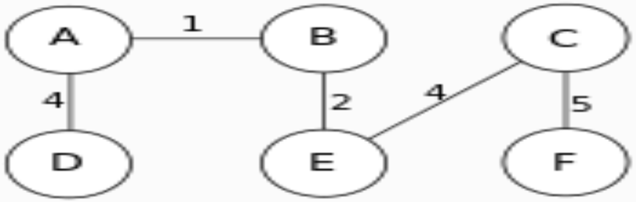
Trees

- **Example 2:**

This figure shows there may be more than one minimum spanning tree in a graph. In the figure, the two trees below the graph are two possibilities of minimum spanning tree of the given graph.

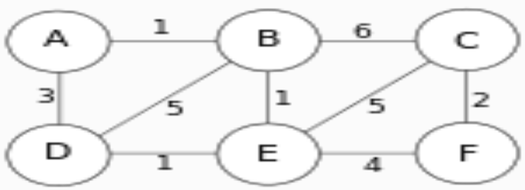


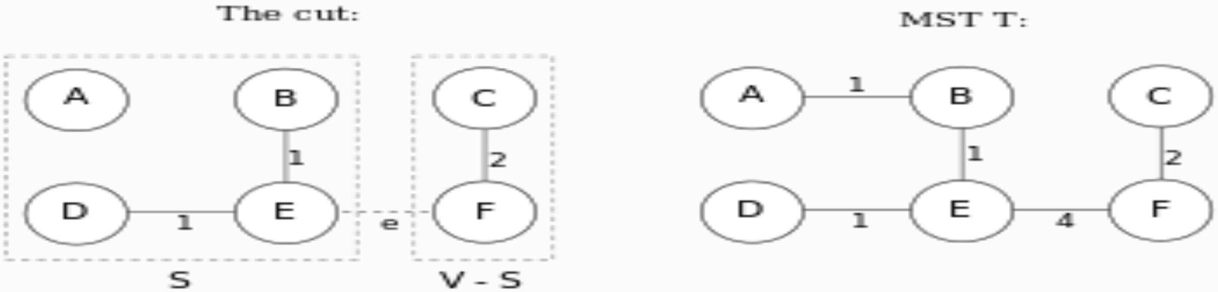




Trees

- *There may be several MST of the same weight having a minimum number of edges*; in particular, if all the edge weights of a given graph are the same, then every spanning tree of that graph is minimum.
- If there are ***n* vertices** in the graph, then each spanning tree has ***n*-1 edges**.





This figure shows the cut property of MSTs. T is the only MST of the given graph. If S = {A,B,D,E}, thus V-S = {C,F}, then there are 3 possibilities of the edge across the cut(S,V-S), they are edges BC, EC, EF of the original graph. Then, e is one of the minimum-weight-edge for the cut, therefore S ∪ {e} is part of the MST T.

Minimum Spanning Tree (MST)

Below are the steps for finding MST using *Kruskal’s algorithm*

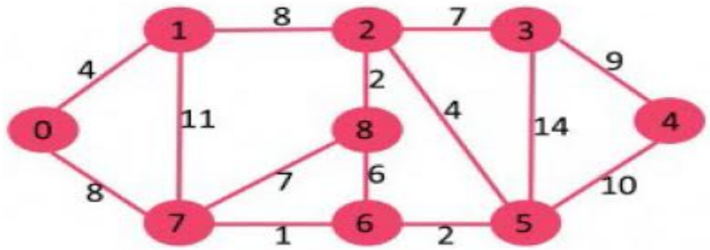
① Sort all the edges in non-decreasing order of their weight.

② Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

③ Repeat step#2 until there are (V-1) edges in the spanning tree.

Trees

- **<u>Example of MST</u>**:



The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

After sorting:

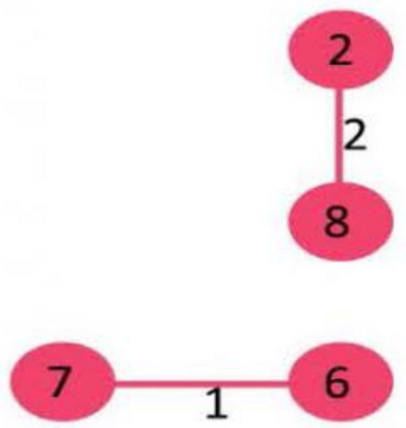
Weight	Src	Dest
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

Trees

Now pick all edges one by one from sorted list of edges

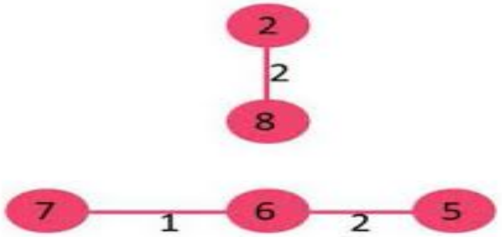
1. *Pick edge 7-6:* No cycle is formed, include it.

2. *Pick edge 8-2:* No cycle is formed, include it.

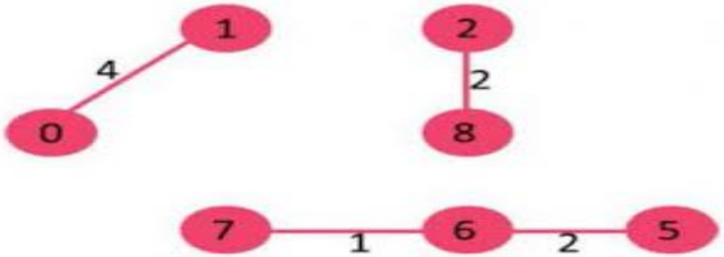


3. *Pick edge 6-5:* No cycle is formed, include it.

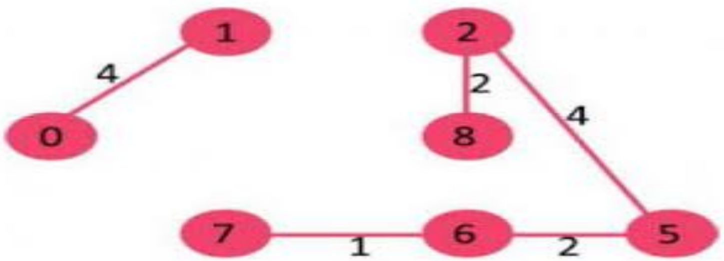
Trees



4. *Pick edge 0-1:* No cycle is formed, include it.



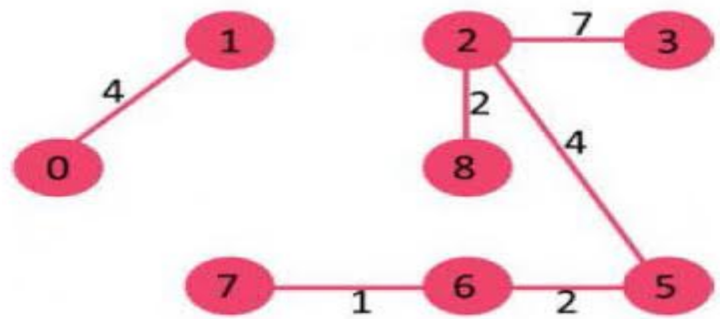
5. *Pick edge 2-5:* No cycle is formed, include it.



6. *Pick edge 8-6:* Since including this edge results in cycle, discard it.

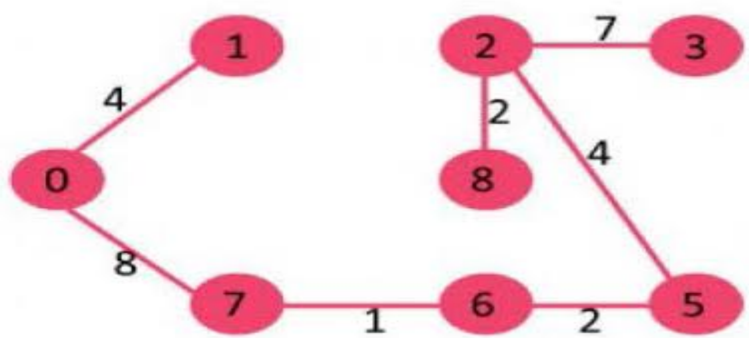
Trees

7. *Pick edge 2-3:* No cycle is formed, include it.



8. *Pick edge 7-8:* Since including this edge results in cycle, discard it.

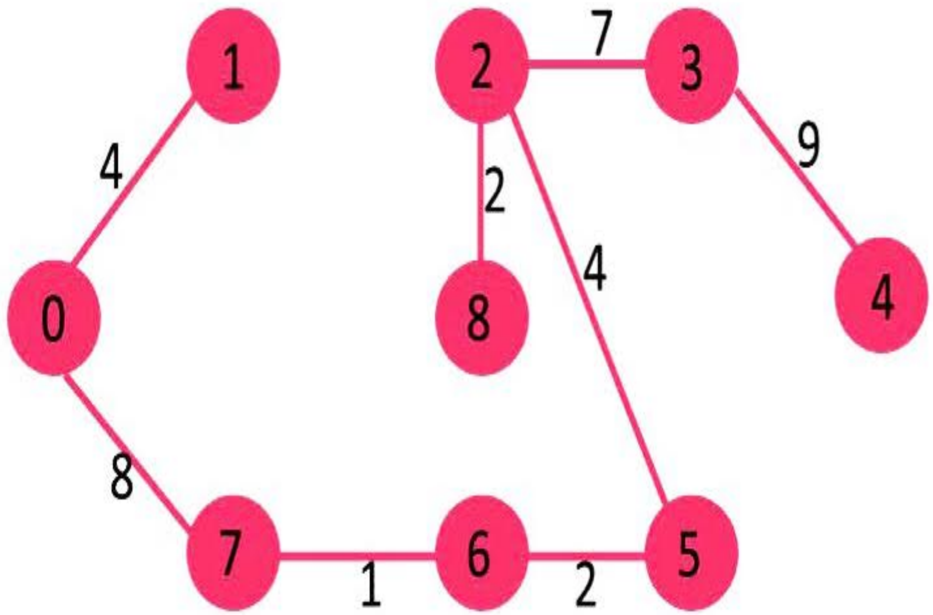
9. *Pick edge 0-7:* No cycle is formed, include it.



10. *Pick edge 1-2:* Since including this edge results in cycle, discard it.

Trees

11. *Pick edge 3-4:* No cycle is formed, include it.



Since the number of edges included equals (V – 1), the algorithm stops here.

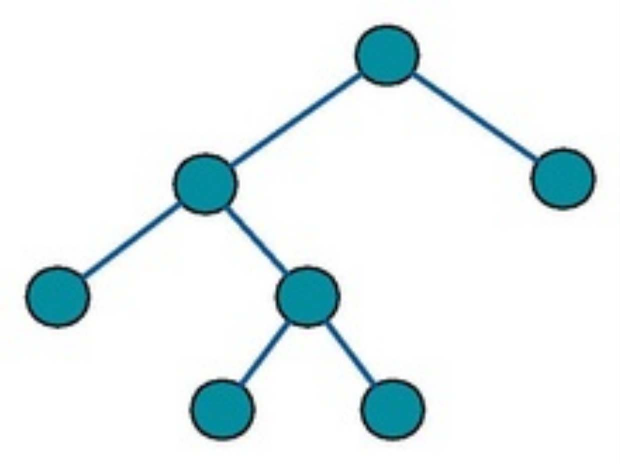
University of Michigan-Dearborn

Binary Trees

Binary Trees

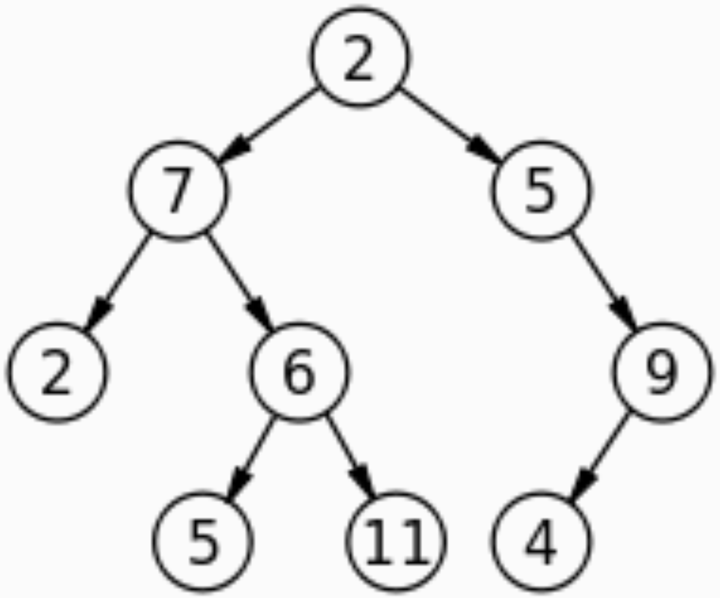
- **Definition:** A *binary tree* is a rooted tree which has either a left child, a right child, a left child and a right child, or no children.
- **Example 1:**

A full binary Tree is a tree in which every node in the tree has either 0 or 2 children.



Trees

- **Example 2:**

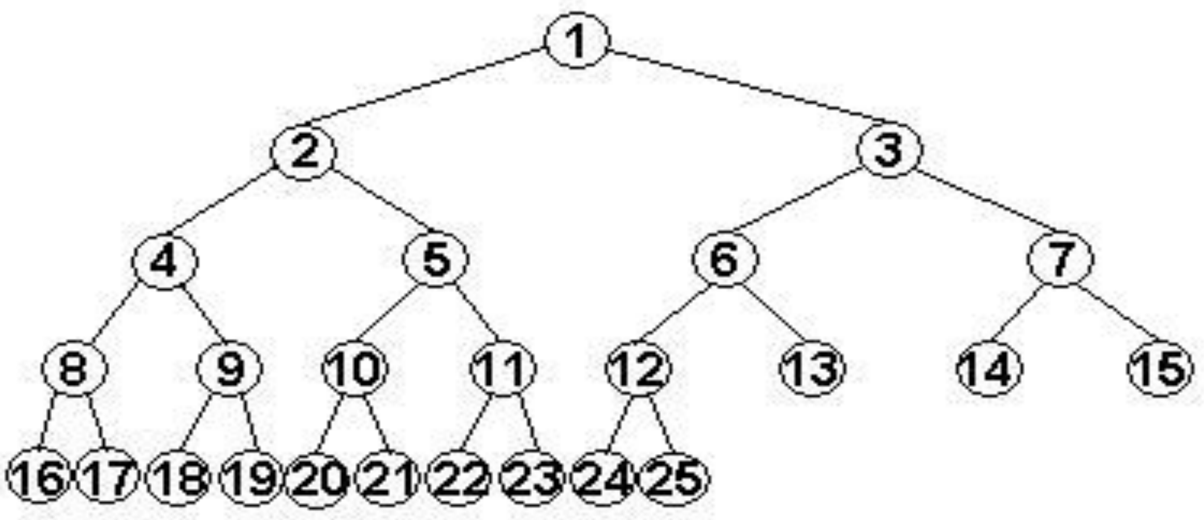


A labeled binary tree of size 9 and height 3, with a root node whose value is 2.

The above tree is unbalanced and not sorted.

Trees

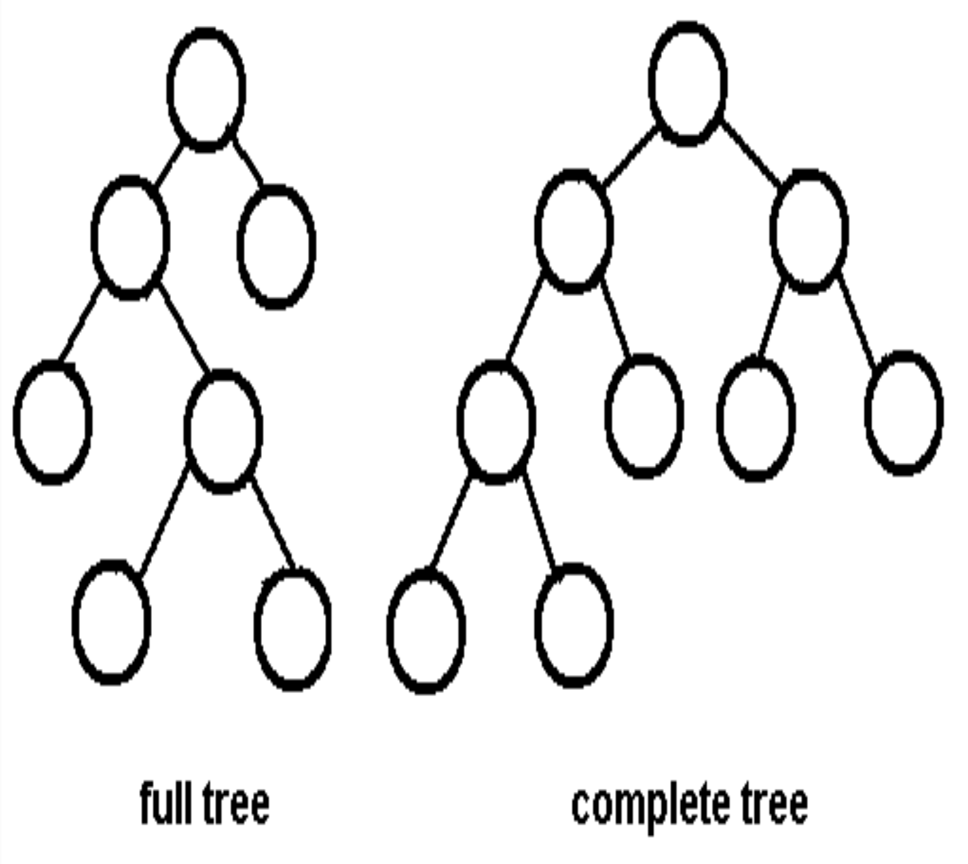
- <u>**Definition:**</u> In a ***complete*** *binary tree* every level, *except possibly the last*, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2*h* nodes at the last level *h.*



A Complete Binary Tree

12 internal nodes, 13 terminal nodes

Trees



Trees

- **Properties of a binary tree:**

1. The maximum number of nodes on level i of a binary tree is 2^i , i >= 0
2. The maximum number of nodes in a binary tree of height k is $2^{(k+1)} - 1$.

   Proof: $\sum_0^k 2^i = \{1- 2^{(k+1)}\} / 1-2 = 2^{(k+1)} - 1$

Trees

DEARBORN

- **Theorem:** If a binary tree of height h has t terminal vertices, then

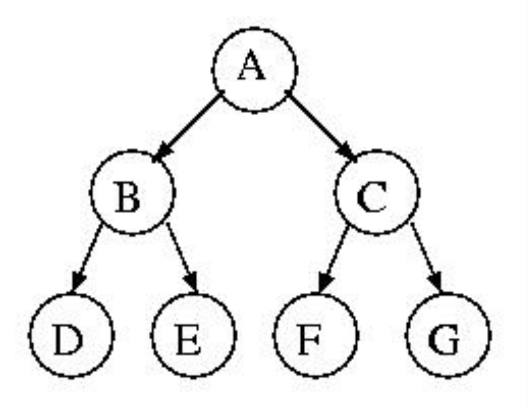
$$\log_2 t <= h$$

- *Proof:*

$$t <= 2^h$$

$$\log_2 t <= h$$

- **Example:**



A Binary Tree of height h = 2 with t = 4 terminals. For this binary tree,

$$\mathbf{\log_2 4 = 2}$$

Trees

- **Definition:** A Binary Search Tree (BST) is a binary tree in which data are associated with the vertices. The data are arranged in which

$$L <= r <= R$$

Where L = the left subtree

r= the root

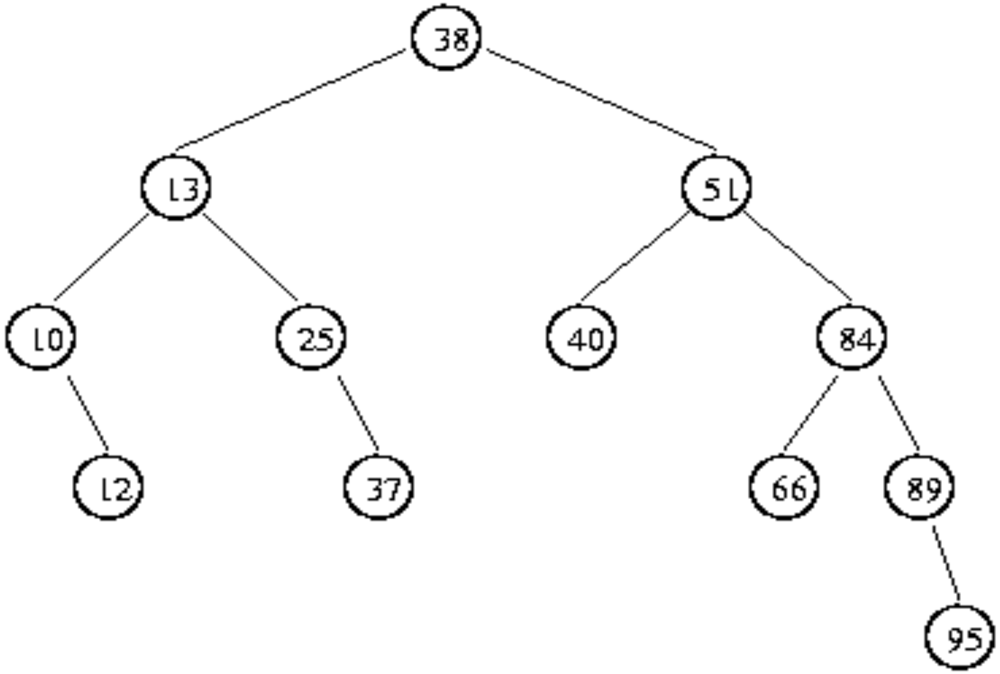
R = the right subtree

Trees

- **Example:**

Binary Search Tree Example

Tree resulting from the following insertions: 38, 13, 51, 10, 12, 40, 84, 25, 89, 37, 66, 95



UNIVERSITY OF MICHIGAN-DEARBORN

Tree Traversals

Trees

- **Definition:** In computer science, *tree traversal* (also known as tree search) is a form of graph traversal and refers to the process of visiting (checking and/or updating) each node in a tree data structure, exactly once.
- Trees can be traversed in 3 orders:

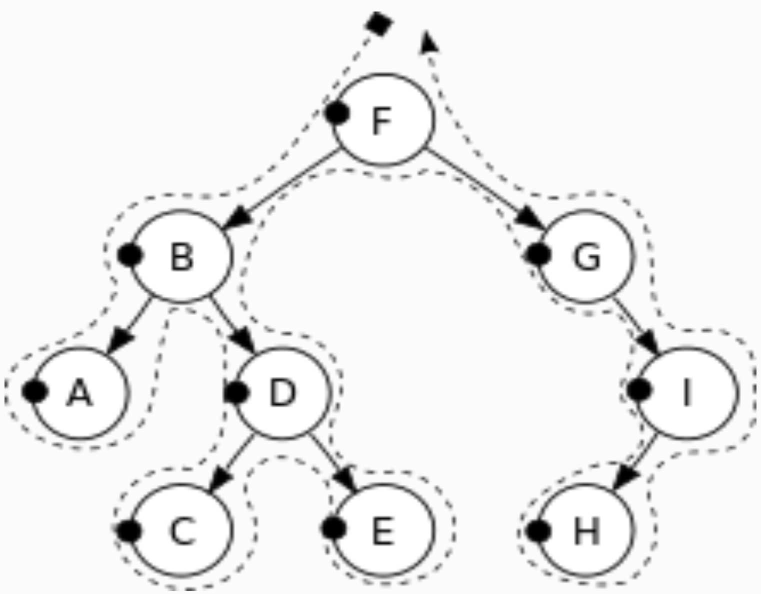
  ① pre-order

  ② in-order

  ③ post-order

Trees

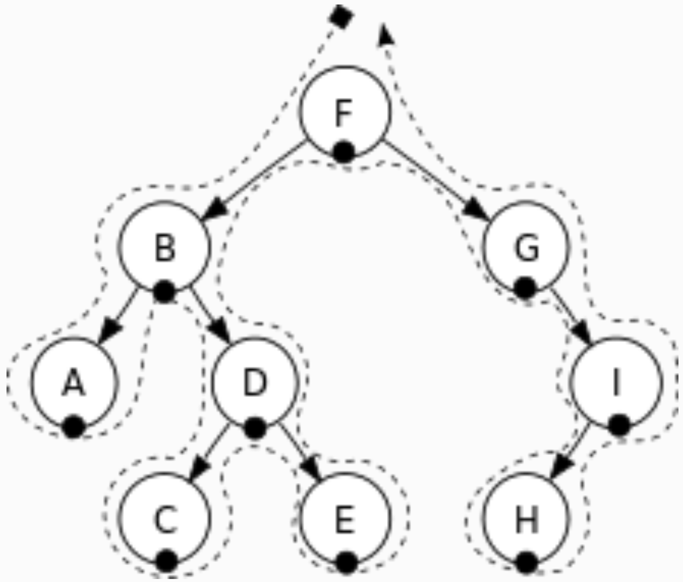
- **Pre-order**
  - Display the data part of the root (or current node).
  - Traverse the left subtree by recursively calling the pre-order function.
  - Traverse the right subtree by recursively calling the pre-order function.



Pre-order: F, B, A, D, C, E, G, I, H.

Trees

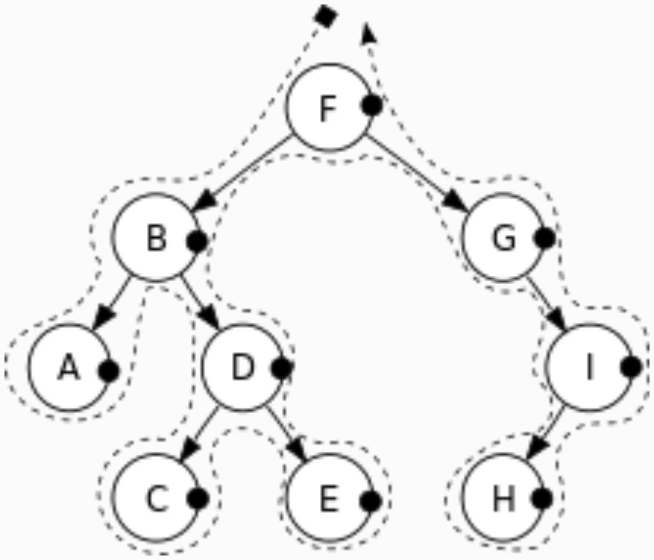
- **In-order**
  - Traverse the left subtree by recursively calling the in-order function.
  - Display the data part of the root (or current node).
  - Traverse the right subtree by recursively calling the in-order function.



In-order: A, B, C, D, E, F, G, H, I.

Trees

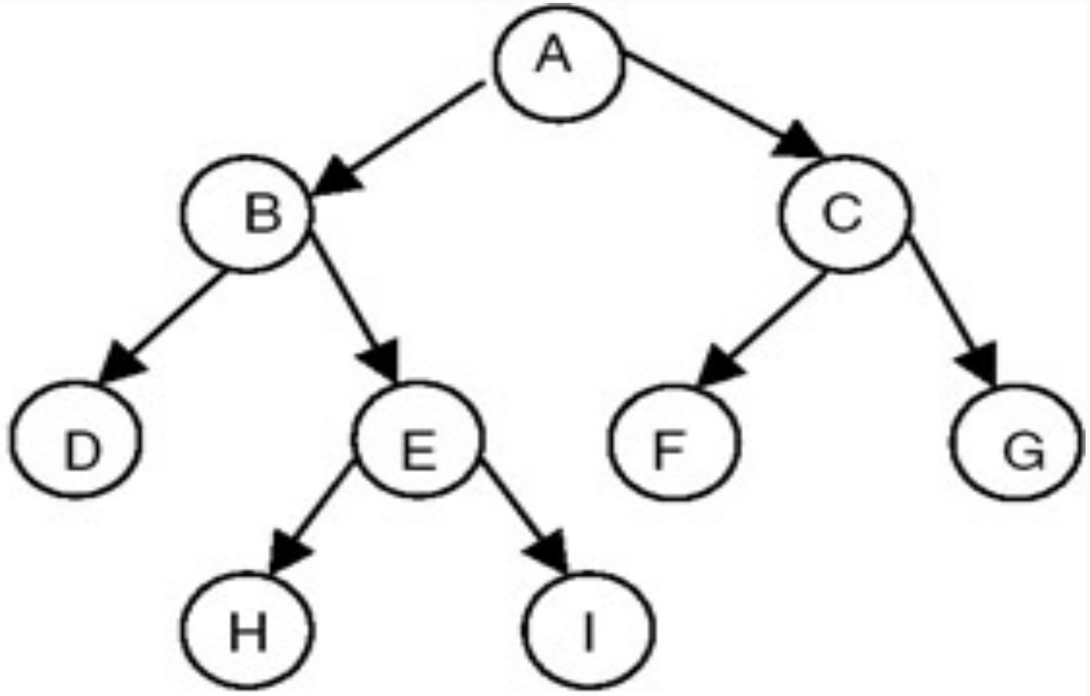
- **Post-order**
  - ➢ Traverse the left subtree by recursively calling the post-order function.
  - ➢ Traverse the right subtree by recursively calling the post-order function.
  - ➢ Display the data part of the root (or current node).



Post-order: A, C, E, D, B, H, I, G, F.

Trees

- **Example 1:** *A binary tree along with its inorder, preorder and postorder.*



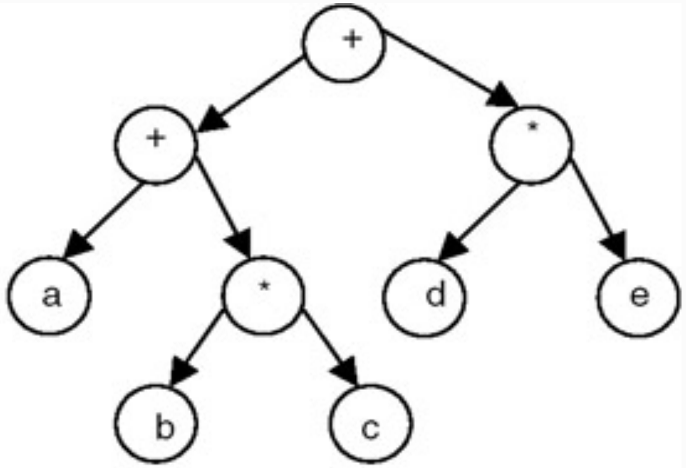
Inorder : DBHEIAFCG

Preorder : ABDEHICFG

Postorder : DHIEBFGCA

Trees

- If an expression is represented as a binary tree, *the inorder traversal of the tree gives us an infix expression*, whereas *the postorder traversal gives us a postfix expression* as shown in this Figure.
- **Example 2:**



Inorder : a + b * c + d * e

postorder : abc*+de*+

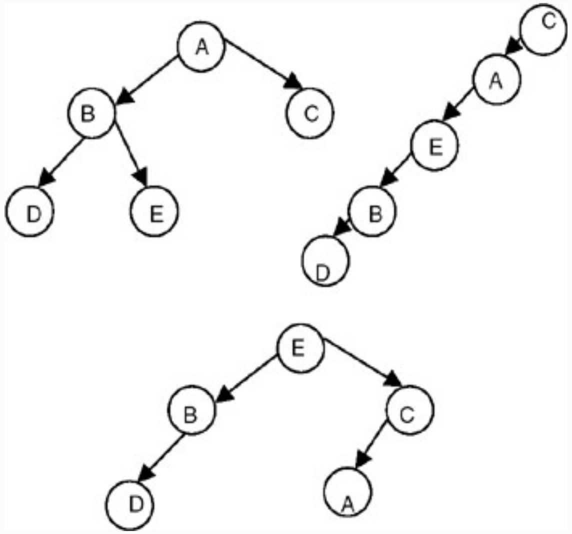
In this figure, a binary tree of an expression along with its inorder and postorder.

Given an order of traversal of a tree, it is possible to construct a tree; for example, consider the folowing order:

Inorder = DBEAC

Trees

- **Example 3:** We can construct the binary trees shown in this figure by using this order of traversal:



Binary trees constructed using the given inorder.

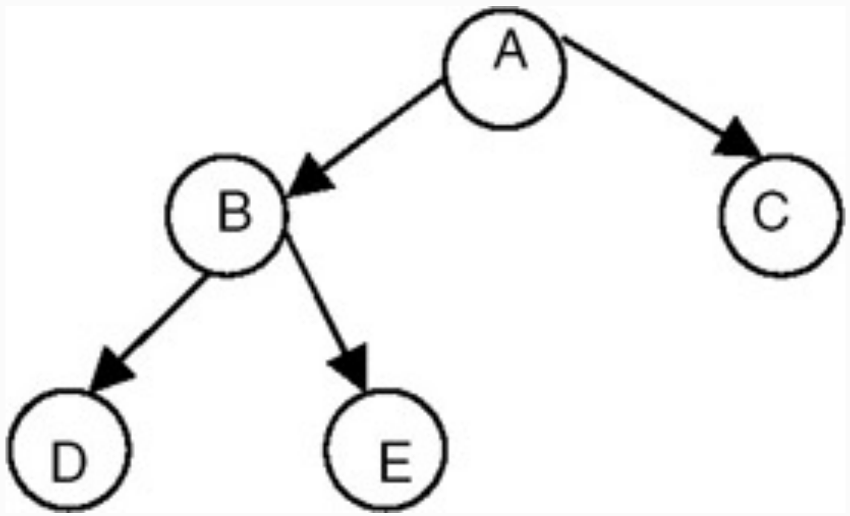
Therefore, we conclude that given only one order of traversal of a tree, it is possible to construct a number of binary trees; a unique binary tree cannot be constructed with only one order of traversal. For construction of a unique binary tree, we require two orders, in which one has to be inorder; the other can be preorder or postorder. For example, consider the following orders:

Inorder = DBEAC

Postorder = DEBCA

Trees

- **<u>Example 4:</u>** We can construct the unique binary tree shown in this Figure by using these orders of traversal:



A unique binary tree constructed using its inorder and postorder.