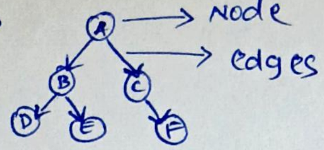
Data structures : (Nov-5th-2021)

Binary Tree :-

→ 

→ Tree Contains many nodes we draw nodes as circles, this nodes can point to other nodes. lines which connect nodes are called edges,

→ We can store values within nodes of tree.

→ B is parent for D and E nodes and also it child for node A like that we can write relationships. So In a tree a node can be parent and child based on context.

→ root :- root is a node that has no parent (In above A is root)

→ leaf nodes :- D, E, F called leaf nodes. and this nodes don't have childrens.

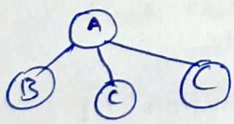
→ In binary we will be having one root node many leaves.

→ In above tree, every leaf is two edges away from the root, by counting number of arrows from the roots to any leaf.

→ ~~leaf~~ leaves may occur different ~~levels~~ levels as well Ex:- 

→ C is one edge away.

Binary Tree :- It's tree every node has atmost two children. above tree is binary tree



→ it is ternary tree as it has 3 childs,

even it has less than two child also it is a binary tree.

Rules for binary tree :- 1) At most 2 childrens per node 2) exactly 1 root.

3) Exactly one path b/w root and any node (lets see path b/w A-E then the only one path is A to B to E. Then And this is only way to get from A to E.

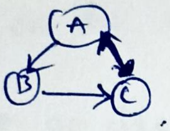
→ queue is empty = Done

→ (A) → still Consider a binary tree

→ when we have no nodes we consider that as empty tree.

→ Any emty tree, we can Consider as a binary tree

Let's take:



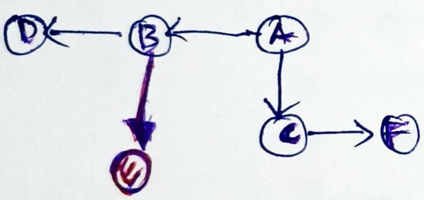
→ At most two clids ✓

→ We don't have root node ✗ (means exactly node without pare

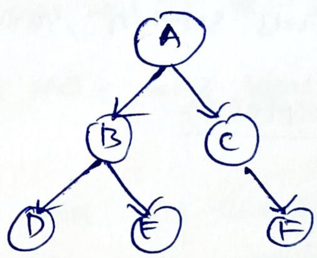
→ ~~are~~ multiple path exist ✗ path for C is 1) A - BC ② A → B → C → A → B →

↳ we can write Infinite pats as it is a Cycle.

→ So in Algorithm problems it maynot be a nice binary tree diagram for ex:-



same as



Binary tree programatically:-

→ we will represent tree as Object. here every node going to be some objects. properties within this object would be the current value. exame A is value of root node

→ we also need to store left and right pointers (node.left & node.right) children with those are properties on that object.

→ if node is having one child ~~or~~ no child → so we use empty values for that child like null/or undefin

DFS:- (we get root node as input) (Assume above tree)

→ In DFS we start with root node A and add it Some Collection and we have to maintain in very particular order. $\boxed{\text{Values: A}}$

→ Then from there ~~or~~ I have to goto B → as it hav: DFS I have to goto D. value: values: A B D

→ After D their is nowhere deeper. so now Iam moved laterally to to the e node. (values: a, b, d, e)

→ Then i don't have any depper from (E) so it goto c (values: a, b, d, e, c)

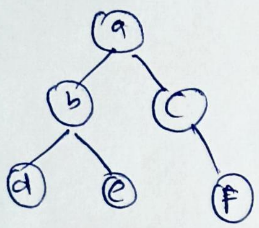
→ from c i goto f (values: a, b, c, d, e, f)

This is DFS on the binary tree.

Basically we go deeper until we found leaf node and then we move acros

→ We will use stack structure here.

~~Algorithm~~



1) take root Node a and store it on stock [a]

2) check stack is <u>empty</u> (right now it's not empty)

current node [a]

3) will remove 'a' from stack and label it as current node

(if something leaves stack i will consider it as visited, because

I need to list out my values as that is my problem.

4) look at that node childrens. and i see that is has b child on it's left and 'c' on it's ~~left~~ right

5) I will push those two childrens on stack [b / c] (first push right first with left child)

6) Now check stack is empty?

Current: [b]

7) Remove top of the stack → [c]

8) Consider b childrens → and add them both [d / e / c]

9) Now check stack is empty

current: [d]

10) Remove top → [e / c]

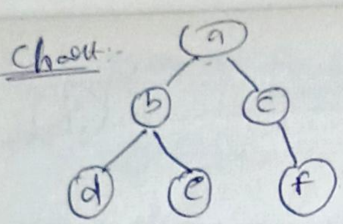
11) look at d children it has no children. so nothing to add to stack. so technically i finished this iteration

12) Now i pop e from stack [c] current: [e]

.... it goes on. until stack is emty that means you travelled entire tree

→ quee is empty = Done

Chart:-



1) root node onto stack

2) left right of root

values: a,

3) left & right of b → values: a, b

4) no childs for d values: a, b, d

5) no, childs for e values: a, b, d, e

6) left & right of c values: a, b, d, e, c

7) no, child for f values: a, b, d, e, c, f

8) stop as stack is empty.

Breadth first values.

→ we use queue here, nodes enter from back of queue and lete front of queue (Enque → Add dequeue = remo)

Step1 → Enquee root $\xrightarrow[\text{queue}]{a}$

Step2 → queue is empty — NO, so Current: a

Step3 → Remove front element of queue ⟶

Step4 → a Check a's Childrey → add to queue $\xrightarrow[\text{queue}]{c,b}$

Step5 ! ⟶ c ⟶ Current: b

Step6 !- add b's childrey $\xrightarrow[\text{queue}]{e\ d\ c}$

Step7 !- c leaves $\xrightarrow{e, d}$ c = current.

Step8 !- add c's Chidsen and rem 'd' $\xrightarrow{f, e}$ Curr: d

Step !- since d has no Childrey nothin to add

e " " "

f " " "

→ quee is empty = Done.