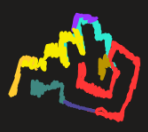
Group project - 3 members / 4 members.



Read DSA Research Papers on Applications of DS.

↳ Implement the abstract idea as project.

DSA application papers IEEE, Springer, libgen, z-lib.

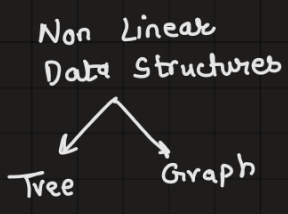
w1 → Team details
w3 → Broad Area ⟶ give the area/choice of your to use DS.
w5 → Project Abstract

man bc

for seeing the functionalities of bc

Datastructures:

→ Arrangement of data in computer's memory in a structured way with some specific operations performable on it.



Non linear as no unique before/after element.

Tree

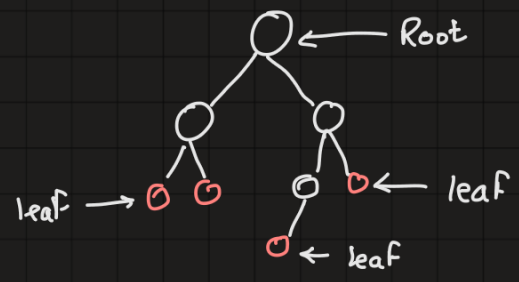
28/02/23

Components :

root , branch, leaf , Node.

→ 0 nodes can be a tree.
→ 1 node can be a tree, with the node being root node.

L-DS	NLDS
data arranged in orderly manner,	!
Data elements can be accessed in one time	!
Simpler implementation	!



Application :

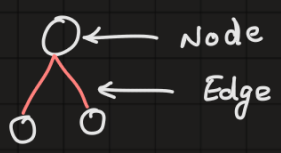
- ★ File system
- ★ Domain Name Server (DNS) Tree
- ★ College info. system
- ★ Decision support system
- ★ Syntax tree (used by compiler)

HW: check 'man tree'

Node - Smallest element of storage in a data structures.

Tree Terminology:

If any tree has 'N' nodes there will 'N-1' edges.

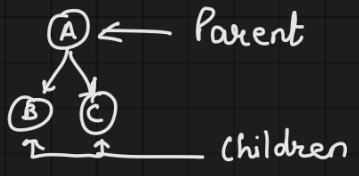


Edge → Link connecting two nodes.

1. Root:
   - First Node is called Root Node.
   - There can be only one root

2. Edge:

3. Parent:
   Node which is predecessor of any other node is it's parent.



4. Child:
   Node descendent of any

5. Sibling:
   Same level, Same parent.

6. Leaf:
   Node with no children; last level.

7. Internal Nodes:
   → Nodes which are not leaf.

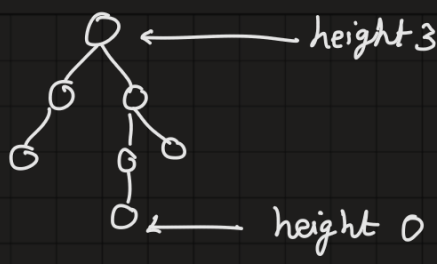
8. Degree of each node:
   → No. of children a node has

9. Degree of tree:
   Maximum degree of tree among all nodes.

10. Level:
    → Root is at level 0.
    → children' of root at lv.1 and so on.

→ each step from top to bottom is called as a Level.

11. Height:
→ Reverse of level.
→ leaf is at height 0.
→ Root is at max height

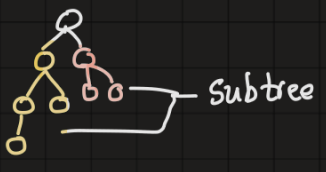


12. Depth
→ Synonymous to Level but depth is for particular Node Level is not.

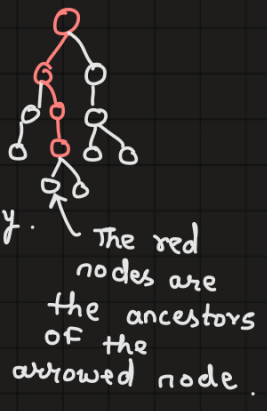
13. Path: Sequence of Nodes and Edges from one node to another is Path.

14. Subtree:
In a tree, each child from a node forms a subtree recursively.

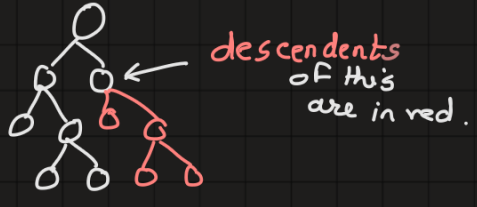
Every child tree forms a subtree on its parent



15. Ancestors:
↳ Parent, grand-parent, etc..
↳ the path from current Node till Root



16. Descendents:
↳ All children and so till the leaf of Family.



Types of Trees.

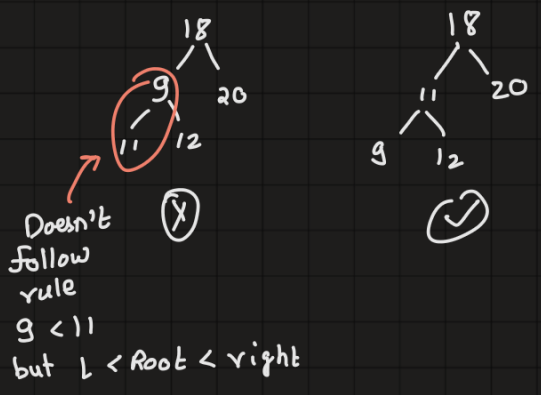
1) Simple Binary Tree :
   → Max 2 children per node.

2) N-ary Tree
   → 'N' max children per node.
      → e.g. 3-ary tree, 4-ary tree.

3. Binary Search Tree.

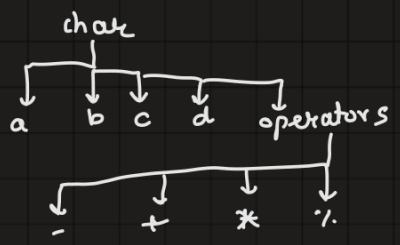
   → All elements in left subtree are less than root.
   → All elements in right subtree are greater than root.

   * All Nodes should follow the rules, not only root.



Types of trees: 03/03/2023

General tree:
→ No constraint is placed on hierarchy of tree.
→ There can be 'n' number of children for every node.



General tree.

e.g. File system directory, there can be as many folders as needed in each directory.

Binary tree:
Max 2 children.

Binary Search tree.
→ Binary tree (Go up for details).

Types of Tree:

AVL tree
↓
Adelson-Velshi-Landis. (Designers

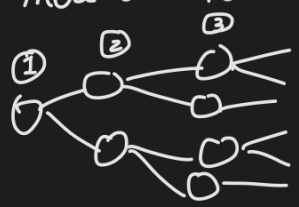
→ AVL tree is a Balanced binary Search Tree.

→ Red black tree:
→ Balanced Binary tree.

Variations of Binary Search tree
→ B-
→ B+ } used in DBMS for indexing.
→ Min-max heaps
→ Splay Trees
→ 2-3 trees.

In DBMS,
Primary keys stored as a tree.

There are levels of index:



① Primary index
② Secondary index
③ Tertiary index.

Binary Tree:

- Children of a node are in ordered pair.
  i.e. we call the children of internal node, left child and right child.

Strict/full binary tree:

→ Every non leaf node/ internal node has exactly two children.

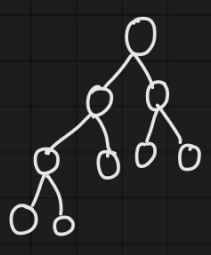
Perfect Binary tree:

→ All leaf nodes are at same level.
→ All internal nodes have degree 2. [i.e. strict/ full binary tree].

→ Skewed binary tree:
→ Every node has 1 child, all on either left or all on right

→ Complete Binary tree.

→ last node might have only one left child



Complete Binary tree.

Complete

↳ Since, all nodes before last are filled.

→ Perfect Binary tree

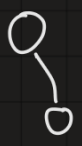
→ 0-2 children
→ All leaf at same level.

complete
skew
full
perfect

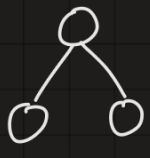
	skewed				
	S-L	SR	FULL	COMP	PERF
1	✓			✓	
2		✓			
3	Just Binary Tree				
4	Just Binary Tree				
5			✓	✓	✓
6			✓	✓	
7			✓	✓	✓

① left skewed
complete

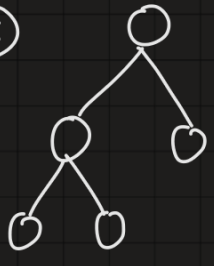
② right skewed



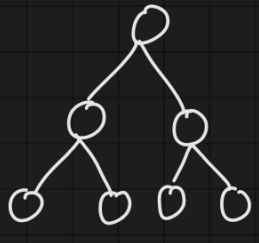
③ ④ binary tree



⑤ Perfect binary tree

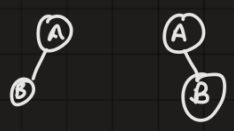


⑥ Complete binary tree



⑦ Perfect Binary tree

General Tree v/s Binary Tree



→ different if binary tree

→ Same if binary tree.

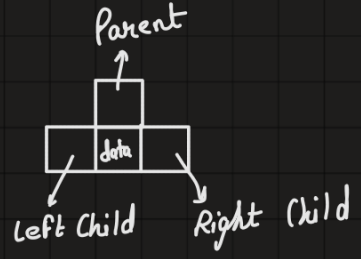
Representation of Tree → How it's going to be stored in computer's memory.

① Linked Representation.

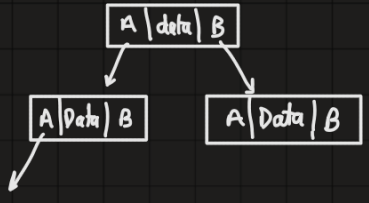
② Array Representation.

① Linked Representation:

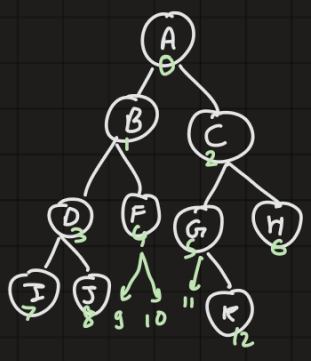
① element (data)
parent
left_child
Right_child
} node address.



② data
left - child
Right - child



③ Implicit Array representation

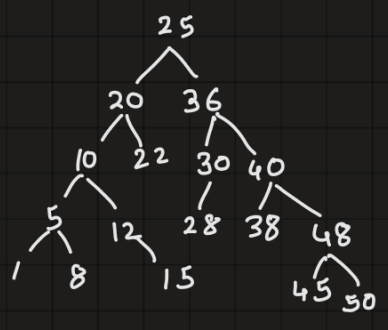


-2^0

-2^1

-2^2

-2^3



$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

25 20 36 10 22 30 40 5 12 - - 28 - 38 48 1 8 - 15 - - - - - - -

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

- - - 45 50

26 27 28 29 30

For finding index of children:

if index of parent = i

child → left = $i \times 2 + 1$

child → right = $i \times 2 + 2$

Number of Nodes in a binary tree, with 'n' levels.

Nodes = $\sum_{i=0}^{n-1} n^i$ OR $\boxed{2^n - 1}$

Finding Parent of child:

$(\text{childIndex} - 1)/2$

① Explicit Array

→ Multidimensional array

data	
Parent	
left	
right	

① How many max nodes can each level 'i' of a bt have?

→ 2^{i-1}

② If there are n nodes in a tree what is the max & min possible height / depth of tree.

③ What will be min/max nodes in a tree with height 'h'?

→ Max → $2^h - 1$

→ Min → $h+1$

→ $Min = \log(n)$

$Max = n - 1$

$\log_2^n$ ⇒ Means : Number of times you keep dividing n till it is not further divisible.

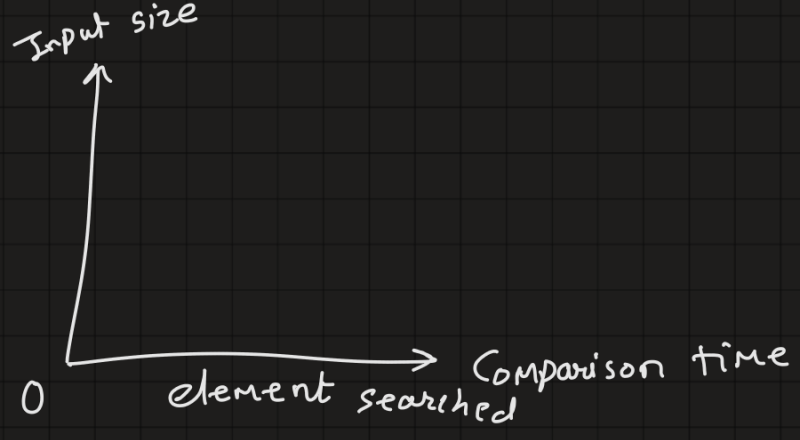
Min	Max
$h+1$	$2^h - 1$

Mathematical derivation of minimum nodes

NULL Tree has height -1

10000 Nodes, random nodes.

1. Create File of random nodes (100000)
2. Insert data in BST.
3. Store number of searches.
4. Plot a graph for
   data found, not found, no. of comparisons



Search same element in multiple sized tree.

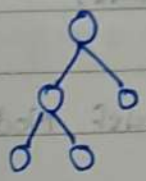
classmate
Date 06/03/23
Page

Binary Search Tree :

→ All nodes are unique.

→ In binary tree data can be repeated.

Binary Tree
~~BST~~ operators:



✓ Insert
- remove (delete)
✓ - Search
✓ - display (traverse)
✓ - isEmpty
- height
- depth
- count Nodes
- ~~levels~~ display Nodes In I^{th} level.
- ~~display~~ type of tree.
- ~~check~~ isBST()
- mirror OfAtree.
- findAncestors/ decendents of a tree.
✓ - count (No of leaf, no. of internal nodes)
- destroy

files:
main.c
tree.c
tree.h

// root is a unique node of a tree & not a tree in itself.

bst.h

```
typedef struct Node {
    int data;
    Struct Node * left , *right;
} Node;

typedef Node * BST;

void initBST( BST *t);
void insert( BST *t, int key);
int search (BST t , int key); // returns 0 or 1
```

bst.c

```
#include "bst.h"
#include "stdlib.h"

void initBST ( BST *t){ {
    *t = NULL;
return;
}

void insert (BST *t, int key){

    Node * nn = ~~mall~~ (Node*) malloc (sizeof (Node));
    if (!nn)
        return;
    nn → data = key;
    nn → left = ~~No~~ nn → right = NULL;
```

```
Node *p, *q = NULL;
p = *t;
if (!p) {
    *t = nn;
    return;
}

while (p) {    q = p;    // preserving p by storing ≠ p's value in q.
    if (p->data == key)
        return;
    if (p -> data < key)
        p = p -> right;
    else
        p = p -> left;
}

if (q->data > key) {
    q -> left = nn;
else
    q -> right = nn;
return;
}

int search( BST t, int key) {
    Node *p = t;
    if (!p)
        return 0;
    while (p) {
        if (p -> data == key)
            return 1;
```

```
        if (p->data >= Key)
            p = p->left;
        else
            p= p->right;
    }
    return 0;
}
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "BST.h"

int main {

    BST t;
    initBST(&t);
```

Date
Page

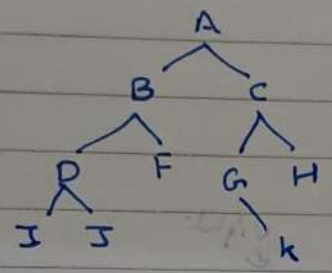
Binary Tree Traversals

6 traversals: L-N-R, L-R-N, N-L-R, N-R-L, R-N-L, R-L-N

- Inorder — LNR
- Preorder — NLR
- Post order — ~~N~~LRN

} 3 accepted Norms

The root decides traversal, root i.e. Node.



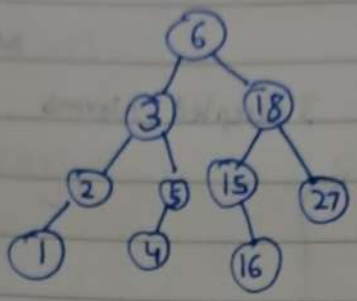
Preorder → Root

Inorder	I D J B F A C G K H	[Left : Node Right]
~~Postorder~~ ~~Preorder~~	I J D F B K G H C A	[Left - Right - Node]
~~Postorder~~ Preorder	A B D I J F C G K H	[Node - Left - Right]

* Left always comes ~~become~~ before right.
* Node is changing.

Math equations as tree:

- All operands as leaf, all operators as internal nodes.



6
↓
3
↓
5 ← ✓

Inorder: LNR : 1 2 3 4 5 6 15 16 18 27
Preorder: NLR : 6 3 2 1 5 4 18 15 16 27
Postorder: LRN : 1, 2, 4, 5, 3, 16, 15, 27, 18, 6

Pre-order to tree construction:

left root right

Inorder: (I) D (J) B F A ~~K~~ ~~K~~ C (H)

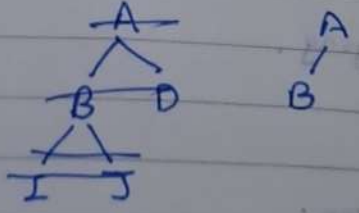
Preorder: A B D I J F C G K H

~~root of~~ NLR

Middle of inorder is root

~~so~~ 2nd element of pre-order is root of subtree-left.

~~3rd element of~~ pre-order is root of subtree-right.



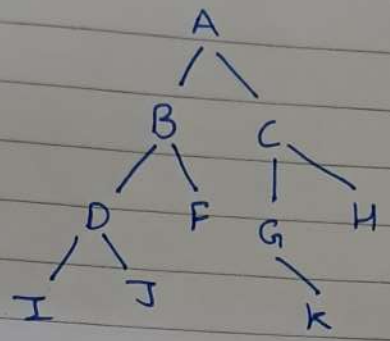
In : I D J B F A G K C H

Pre : A B D I J F C G K H

(A = root)

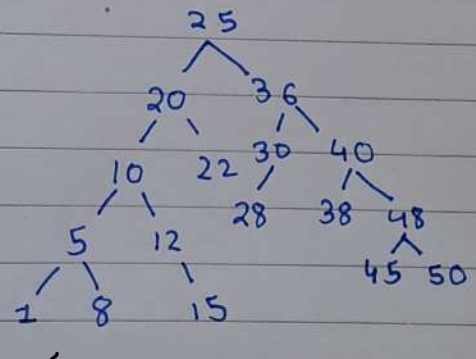
LNR

NLR



Selection of Nodes:

1. A root
2. B
3. D
4. I left of D
5. J right of D
6. F right of B
7. C right of A
8. G left of C
9. K right of G
10. H right of C.



Inorder: 1,5,8,10,12,15,20,22,25,28,30,36,38,40,45,48,50

Preorder: 25, 20, 10, 5, 1, 8, 12, 15, 22, 36, 30, 28, 40, 38, 48, 45, 50

Postorder: 1, 8, 5, 15, 12, 10, 22, 20, 28, 30, 38, 45, 50, 48, 40, 36, 25

e.g.

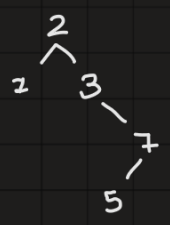
Inorder: 1 (2) 3 5 7

Preorder: 2 1 3 7 5

Postorder: 1 3 7 5 2

Steps:

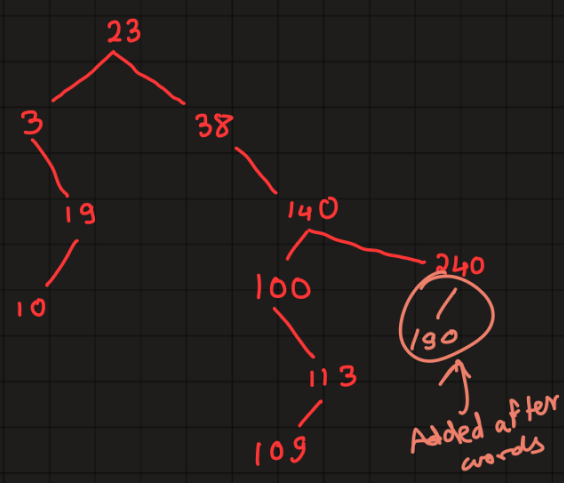
① Preorder's first node is root
② left of '2' in inorder is left sub-tree
right of '2' -11- right -1-
③ right of 2 is 3. (from Preorder)
④ Similarly, 7 right of 3
⑤ 5 is left of 7.



Functions for display:

① InOrder

```
InOrder ( BST t){
  if (!t)
    return;

  InOrder ( t->left);
  printf("%d  ", t->data);
  InOrder (t->right);
return;
}
```



Input Tree

/*

LNR InOrder : 3,10,19,23,38,100,109,113,140,240.
NLR PreOrder : 23, 3, 19, 10, 38, 140, 100, 113, 109, 240
LRN PostOrder : 10, 19, 3, 109, 113, 100, 240, 140, 38, 23

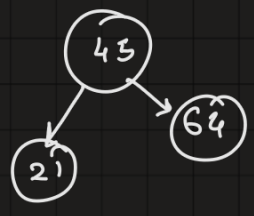
Binary Tree creation:

Pick a number from File of 10000 nodes.

generate random number:
if No. is odd attach to left
else attach to right.

traverse till you reach leaf and then attach.
keep random generating till you reach leaf.

e.g. 45
64
39
21
18
3
27
20



For Gen random,
64 32 → right ∵ 32 is Even
39 33 left
21 31, left, again generate till not at leave.

/*
Gen random

Sudo Code

main( ) {
open file
start reading
while( )
insertInBT(BT *t , int data);

}

Assignment 2

```
void insertInBT (BT *t , int key){
    Node *nn = (Node *) malloc (sizeof (Node));
    nn->left = nn->right = NULL;
    nn->data = key;

  if(!t){
     t = nn;
     return;
  }
  Node *p = *t;
  Node *q;

  while (p) {
     q = p;
  if(gen() %. 1  == 0) {
```

```
void insertBST(BST *t, int Key){
    if(!t){
        t= createNewNode(Key);
        return;
    }

    if(t->data== Key)
        return;
    else if(t->data > Key){
        if(t->left){
            insertBST(t->left,Key);
        else
            t->left = createNewNode(Key);
        }
        return;
    } else {
        if(t->right){
            insertBST(t->right);
        else
            t->right = createNewNode(Key);
        }
        return;
    }
```

slow

slow

easier function:

```
if(!t->left)
    insert(t->left,Key);
```

```
if(!t->right)
    insertBST(
    t->right,Key);
```

Testcase : 10

Tree :

HW

countNoOfNodes ( )
CountLeaf( )
CountInternalNodes ( )

Check whether tree is free (Every Node has 0, 2) children.
Find height of tree ( )

```
int CountNodes (BST t) {
    if(!t)
        return 0;

    return 1 + countNodes(t->left) + countNodes(t->right);
}
```

using while & linked stack →

```
int countNodes (BST t) {
    if(!t)
        return 0;
    int count = 0;
    Node *p = t;
    Queue q;
    initQueue(&q);
    enqueue(&q, t);
    while ( !isEmpty(t)) {
        p = deQueue(t);
        count ++;
        if (p->left)
            enqueue (&q, p->left);
        if (p->right)
            enqueue (&q, p->right);
    }
    return count;
}
```

using Queue :

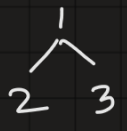
Write code using stack

```
int countNodes(BST t){
if (!t)
    return 0;

Node *p = t;

Stack S;
initStack(&S);

push(&s, p);
while (!isEmpty(s)){
    p = pop(&s)
    count++;
    if(p->left)
        push(&s,p);
    if (p->right)
        push(&s, p);
}
```

```
int countLeaves(BST t){
if(!t)
    return 0;

if( !t->left && !t->right)
    return 1;

return countLeaves(t->left) + countLeaves(t->right);
}
```

IsFull Tree using Multiplication.

```
int isFullTree(BST t){
    if(!t)
        return 1;

    if(t->left && t->right)
        return isFullTree(t->left) * isFullTree(t->right);
else if( !t->left && !t->right){
        return 1;

    else
        return 0;
}
```



```
int max(int a, int b) {
    return a > b ? a : b;
}
```

```
int height(BST t){
    if(!t)
        return -1;          // ∵ height of NULL tree is -1;
    return 1 + max(height(t->left), height(t->right));
}
```

Try it out. →

Homework:

Implement all recursive codes to non recursive

Q ⇒ Check whether tree is complete.

Q ⇒ Find mirror image of tree: {Binary Tree}

Q → Destroy the tree:

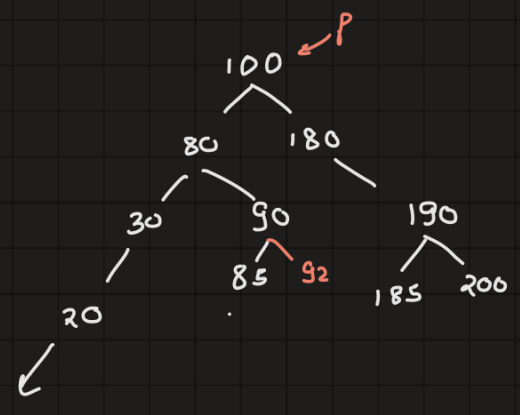
Non-recursive Codes for:
three traversals.

* One tree operation of tree.

Non-recursive code for in-order traversal:

→

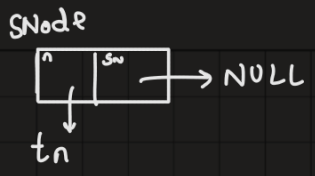
```
if(!t)
    return;
if(!t->left && !t->right) {
    print(t->data);
    return;
}
Node *p = *t;
while(1){
    while(p) {
        push(&s, p);
        p = p->left;
    }
if(isEmpty(s))
    break;
p = pop(&s);
print(p->data);
p = p->right;
}
```




```
typedef struct Stack {
    int top;
    Node ** arr;
    int size;
}
```

```
typedef struct StackNode {
    Node* n;
    struct StackNode next;
}
```

```
typedef StackNode * stack;
```




```
Node * pop (Stack *s) {
    Node * ret = (*s)->n;
StackNode *temp = *s;
    *s = temp->next;
    free (temp)
  return ret;
}
```

if(!(*s))
return NULL;

Non-recursive
Pre-order,
Postorder,
leaf,
height

15/03/2023

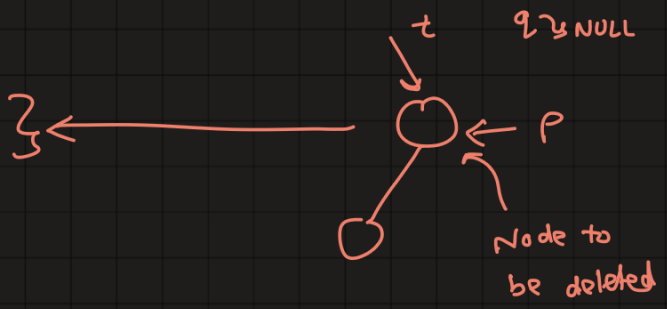
```
void deleteNode (BST *t, int key) {
    if(!t)
        return
```

```
/* Steps:  check for:
        1: empty tree
        2. Search For key
            → Not Found
                return
            Found
        3 leaf Node
        4. Node with left or right child
        5. both children
        6. (3,4,5) && root   */
```

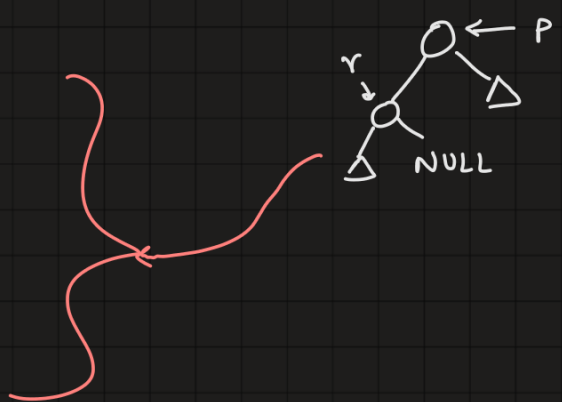
```
if(!(*t))
    return;

Node *p = *t;

Node *q = NULL;

while (p) {
    if(p->data == key) {
        break;
    }else if( p->data > key){
        q = p;
        p = p->left
    }else {
        q = p;
        p = p->right ;
    }
}
if (!p)
    return;
```

```
if(!p->left && !p->right) {
    if(q == NULL) {
        *t = NULL;
        free (p);
    }else if (q->left == p) {          // leaf Node Condition
        free (p);
        q->left = NULL;
    }else {
        free (p)
        q->right = NULL;
    }
    return;
}
```

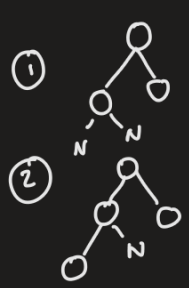
```
// Node with 1 child

if (p->left && !p->right) {
    if (!q) {
        *t = p->left;
    } else if (q->left == p)
        q->left = p->left;
    else
        q->right = p->left
    free(p)
    return;
}

if (!p->left && p->right) {
    if (!q) {
        *t = p->right;
    } else if (q->left == p) {
        q->left = p->right;
    } else
        q->right = p->right;
    }
    free(p);
    return;
}
```

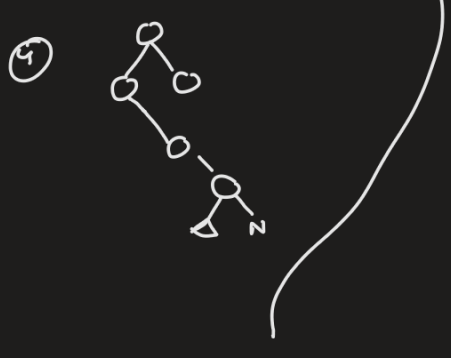



```
Node *r = p->left;
if (!r->right) {
    p->left = r->left;
    p->data = r->data;
    free(r);
    return;
}
```

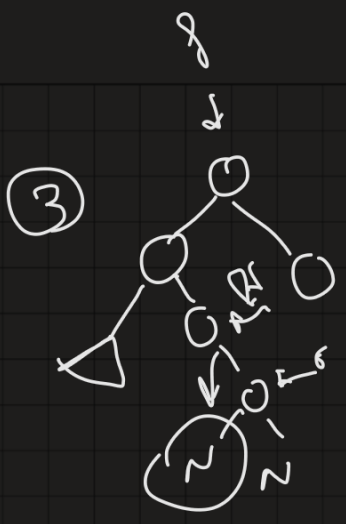


Substitution by Predecessor:





```
Node * parent = NULL;

while (r->right) {
    parent = r;
    r = r->right;
}

parent = r->left;
p->data = r->data;
free(r);
return;
}
```



HW
recursively call delete
for delete(node->left, key);

Tree using Array

```
typedef struct Node {
    int * arr;
    int s;
} Node;
```

definition

calloc → continuous allocation → (void *) calloc (n, size)
- n → no. of elements
- size → size per node in bytes

↳ initializes each block with 0.

```
typedef node * aBST;
void initaBST(aBST *t);
void insert(aBST *t, int key);
void infix(aBST t);
```

arrayBST.c →

```
void initaBST(aBST *t, int key) {
    (*t)->A = (int *) malloc(sizeof(int))
    (*t)->A[0] = INT_MIN;
    (*t)->size = 1;
}
```

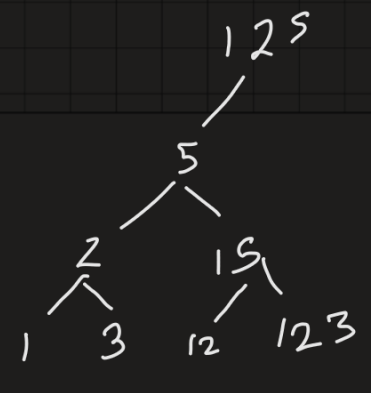
```
void insert(aBST *t, int k){
    if ((*t)->A[0] == INT_MIN){
        (*t)->A[0] = k;
        return;
    }
    int p = 0;

    while(p < (*t)->s && (*t)->A[p] != INT_MIN){
        if((*t)->A[p] == key)
            return;
        else if((*t)->A[p] > key)
            p = 2*p+1;
        else
            p = 2*p+2;
    }

    if((*t)->A[p] == INT_MIN){
        (*t)->A[p] = k;
        return;
    else {
```

BST impl with array:

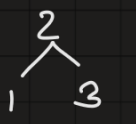
```
typedef struct aBST {
    int *arr;
    int size;
} aBST;

void init(aBST &t) {
    t->arr = NULL
    t->size = 0;
}

void insert(aBST *t, int k);
void inorder(aBST t);

Void insert(aBST &t, int key){
    if (t->s == 0){
        t->arr = (int *) malloc(sizeof(int));
        if(!t->arr)
            return;
        t->arr[0] = key;
        t->s++;
        return;
    }

int p = 0;

while(p < t->s && t->arr[p] != -1){
        if(t->arr[p] == key)
            return;
        if(t->arr[p] > key){
            p = 2*p + 1;
        else
            p = 2*p + 2;
    }
```

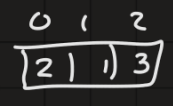
Tree is empty
malloc,
insert at '0'
increase size by 1

check if current is empty
if -1, break.



```
if (p < t->s && (t) -> arr [p] == INT_MIN) {
        t -> arr [p] = key;
        return;
    }

    t -> arr = realloc (t->arr, (p+1) * sizeof (int));
    if (! t -> arr)
        return;

    for(int i = s ; i < p ; i++) {
        t -> arr [i] = int_min;

    }
    t -> arr [p] = key;
    t -> s = p+1;

  return;
}
```






```
void inorderImpl (aBST t , int index){
    if ( index >= t . size || t . arr [index] == INT_MIN)
        return;
    inorder Impl (t, (2 * index + 1));
    if (t . arr [index] != INT_MIN );
    printf (" %d " , t.arr[index]);
    inorder Impl (t ,(2 * index + 2));
  return;
}

void inorder ( aBST t) {
        int index = 0;
        inorderImpl ( t , index);
    return;
}
```

IMPORTANT

this makes sure that we don't access children of INT_MIN(NULL)

Search
Binary Tree Complexity

O( )

	Linked			Array		
	Best	Avg	Worst	Best	Avg	Worst
Insertion	O(log n)	O(log n)	O(n)	O(log n)	O(log n)	O(n)
Delete	1	n	n	1	n	n
Search	1	n	n	1	n	n
Traversal	n	n	n	n	n	n
Height	log n	⟷	n	log n	n	n

→ Avg is n

→ Avg is n as Avg takes n/2 Search time
1/2 is const
∴ O(n)

⟷ No confirm ans, to be proven by experimentation.

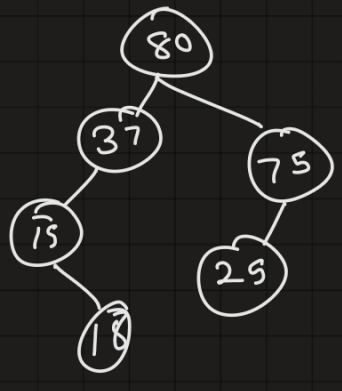
Q. What is number of distinct binary trees possible which represent the array {613, 920, -421, 532, 408}.

→ 42

613

80 37 15 18 75 25 — NLR

15 18 37 25, 75, 80 — LRN



Mirror:

```
if (!node)
    return
else
    mirror (left)
    mirror(right)
    swap (left, right);
```

Postorder

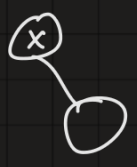
① check NULL ; ret

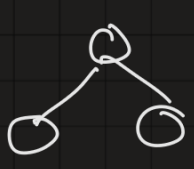
② do {

③ while (not Empty)

Post order:

```
null
return;          xp = root

create stack

do {
   while ( p ) {
         if(p→ right)
              push (p→right);
          push(p)
    }  p=p→left;
     p=pop (&s);

   if ( p→right && p→right == peek(&s)) {
            pop (&s)
            push (p)
            p=p→righ;

     }

    else
      {
          printf("%d", root→data);
          root = NULL   <- IMP * to skip
                           first while loop.
       }

}
```





PS

These are my class notes.
Empty pages indicate I was absent
or highly confused/sleepy.
Read relevant topics from book for the same.

also : [find my codes here]
(github.com/PratyayDhond/DSA-2)