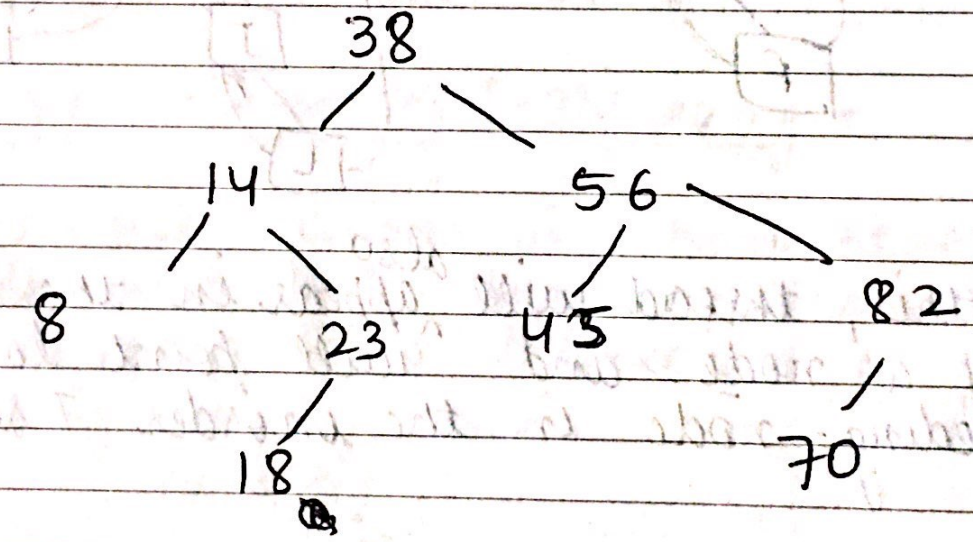
MONDAY

* BINARY SEARCH TREES

Suppose T is a binary tree.
Then T is called a binary search tree if each node N of T has the following properdy :-

The value at N is greater than every value in the left subtree of N and is less than every value in the right subtree of N.

eg



ITEM = 20 Inserted.

Insert 20 to the right of 18

eg 40, 60, 50, 33, 55, 11

THINGS TO DO

Create a Binary Search Tree

MOTIVATIONS

TEAMWORK

Talent wins games,
but Teamwork and
intelligence wins Champions.

MEMO

40
40 \ 60
40 \ 60 / 50
40 / 33 \ 60 / 50
40 / 33 \ 60 / 50 \ 55

40 / 33 \ 60 → 40 / 33 / 11 \ 60 / 50 \ 55

FIND (INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)

A binary search tree T is in memory and an ITEM of information is given.
This procedure finds the location LOC of ITEM in T and also the location PAR of the parent of ITEM. There are three special cases:-

(i) LOC = NULL and PAR = NULL will indicate that the tree is empty.
(ii) LOC ≠ NULL and PAR = NULL will indicate that ITEM is the root of T.
(iii) LOC = NULL and PAR ≠ NULL will indicate that ITEM is not in T and can be added to T as a child of the node N with location PAR.

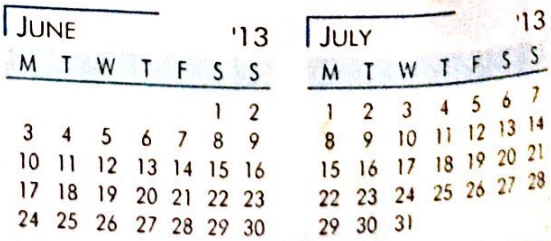
(1) [Tree empty?]
If ROOT = NULL then set LOC := NULL and
PAR := NULL and Return.

(2) [ITEM at root?]
If ITEM = INFO[ROOT] then set LOC := ROOT
and PAR := NULL
and Return

(3) [Initialize pointers PTR and SAVE]

THINGS TO DO

If ITEM < INFO[ROOT]
then :-



Set PTR := LEFT [ROOT] and SAVE := ROOT

Else

Set PTR := RIGHT [ROOT] and SAVE := ROOT

[End of If structure]

(4) Repeat steps 5 and 6 while PTR ≠ NULL

(5) [ITEM found?]
If ITEM = INFO [PTR] then set LOC := PTR and PAR := SAVE and Return

(6) If ITEM < INFO [PTR] then :-
set SAVE := PTR and PTR := LEFT [PTR]

Else :
Set SAVE := PTR and PTR := RIGHT [PTR]

[End of If structure]

[End of step 4 loop]

(7) [Search unsuccessful] Set LOC := NULL and PAR := SAVE

(8) Exit.

IMPORTANT

* Insertion into Binary Search Tree

INSBST (INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM, LOC)

A binary search-tree T is in memory and an ITEM of information is given. This algo finds the location LOC of ITEM in T or adds ITEM as a new node in T at loc LOC.

(1) Call FIND (INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)

(2) If LOC = NULL and Exit.

(3) [Copy ITEM into new node in AVAIL list]

a) If AVAIL = NULL then write: OVERFLOW and Exit.

b) Set NEW := AVAIL, AVAIL := LEFT[AVAIL] and INFO[NEW] := ITEM

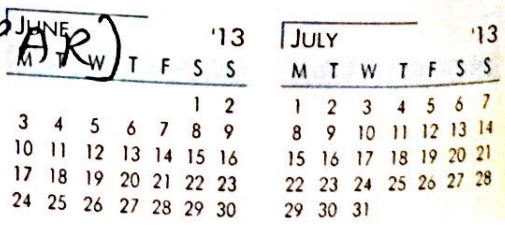
c) Set LOC := NEW, LEFT[NEW] := NULL and RIGHT[NEW] := NULL.

(4) [Add ITEM to tree]

If PAR = NULL then:
Set ROOT := NEW

THINGS TO DO

Else if ITEM < INFO[PAR] then:



Set LEFT[PAR] := NEW

Else:

Set RIGHT[PAR] := NEW

[End of If structure]

(5) Exit.

* Complexity :- $O(\log_2 n)$

* Application of Binary search Trees :-

Delete duplicate elements from list.

If any element is present twice or thrice in a list of elements
Removing redundancy is the application of BST.

* Deletion in BST :-

Suppose T is a BST and ITEM of information is given

Deletion algorithm first find the location of the node N which contains ITEM and also the location of the parent node P(N)

The way N is deleted from the tree depends on no. of children of node N.

AUGUST '13

M	T	W	T	F	S	S
			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
26	27	28	29	30	31	

SEPTEMBER '13

M	T	W	T	F	S	S
30						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	26	27	28	29

THINGS TO DO

There are 3 cases :-

Case 1 N has no children.

Then N is deleted from T by simply replacing the location of N in the parent node P(N) by the null pointer.

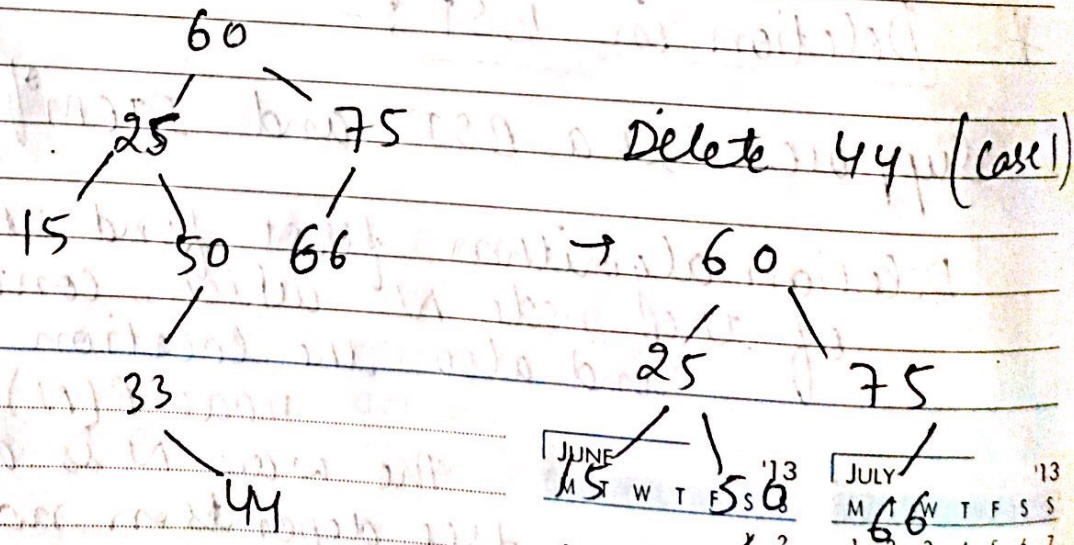
Case 2 N has exactly one child.

Then N is deleted from T by simply replacing the location of N in P(N) by the location of the only child of N.

Case 3 N has two children.

Let S(N) denote the inorder successor of N. Then N is deleted from T by first deleting S(N) from T (by using case 1 or case 2) and then replacing node N in T by the node S(N).

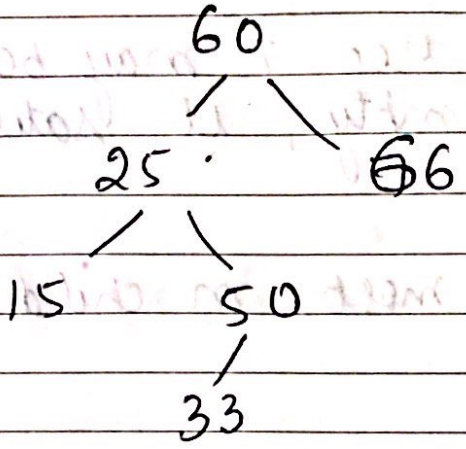
eg (1)



07 SUNDAY

THINGS TO DO

(2) Delete 75 (case 2)



(3) Delete 25 (case 3) (having 2 child)

Find inorder Successor of 25 :-

Inorder Traversal :- 15 25 (33) 50 60 66

50 33 will replace 25

And tree will be :-