1. ~~Linear~~

Linear Data Structure	Non-Linear data structures
→ data elements are arranged in linear order	→ data elements are attached in hierarchical manner
→ Memory is not utilized in efficient way	→ Memory is utilized in efficient way
→ Single level is involved	→ Multiple level are involved
→ eg:- Array, Stack etc	→ eg:-Tree, graph etc

Both Linear and non-linear data structures are non-primitive, that is they are more sophiscated than primitive D.S (int, float, char etc)

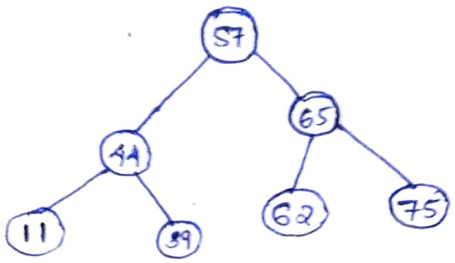
2. ① push(value) → for inserting a value into stack
   → check if the stack is complete (full) [top == size]
   → if yes, the terminate the operation
   → if no, then increment top value by one and set Stack[top] to value (Stack[top] = value)

   ② pop() → to delete a value from stack
   → check whether the stack is empty
   → if yes, terminate the operation
   → if no, then delete stack[top] and decrement top value by one

   ③ display() → to display elements of stack
   → check whether stack is empty
   → if empty, display stack is empty and terminate

operation
→ If no, then define a variable 'i' and initialize with top display stack[i] value and decrement 'i' value by one. Repeat this
- step until 'i' becomes 0.

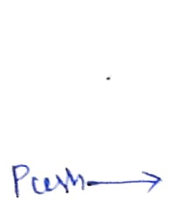
(3) Binary Search Tree.



→ Binary search tree exhibits a has a hierarchical structure
→ Maximum child nodes is two
→ It has a specific order for arrangement of data elements.
→ Since binary search tree are sorted binary trees, it provides faster insertion, deletion and traversal
→ A node's left child must have a value less than its parent's value and node's right child must have a value greater than its parent value.

(4). In accounting method each operations is assigned a charge [more or less than actual cost] called amortized cost.

Consider performing push operation in a stack as



(push operation)
For each step, we consider to give an extra 1 cost.

Therefore actual cost = 5
but, the Amortized Cost = Actual Cost + Credit (cannot be -ve)
= 5 + 5 = 10

5 Disjoint set data structure

A disjoint set datastructure maintains a collection $S=\{S_1, S_2 \ldots S_k\}$ of disjoint dynamic set.

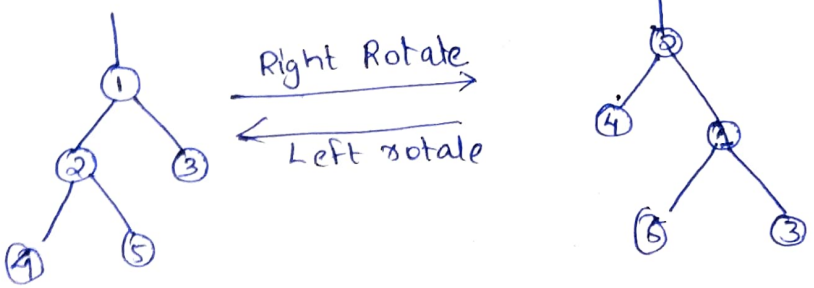
$S_x = \{1, 2, 3, 4\}$
$S_y = \{11, 12, 13, 14\}$
} Since there is no common elements in both set, they are disjoint set

Various disjoint set operations are

1. Make-Set(x).
2. Union(x, y)
3. Find Set(x)

6. Types of rotation

Basic operation of changing tree structure is called rotation



7. Red black tree

Red black is a type of binary search tree, where every node is dosed withe Red or black

Properties

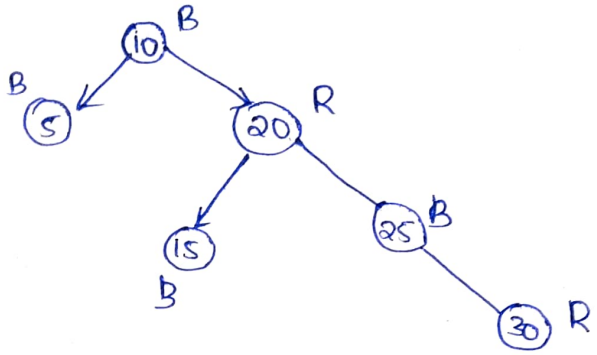
→ Root node is black coloured

→ Chioldrens of Rednode must be black

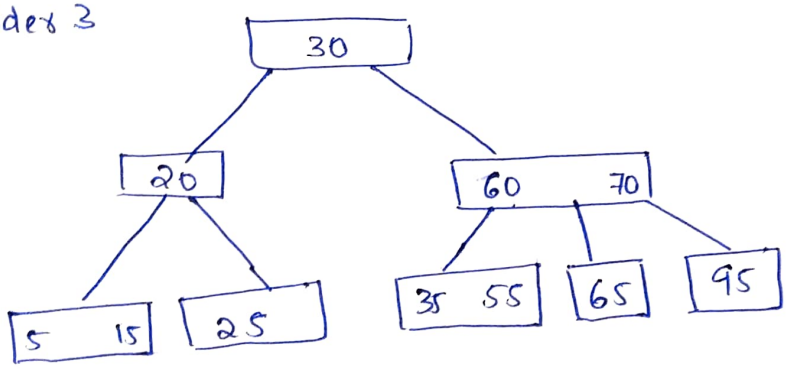
→ Red colour is used for each new node insesk

→ In the tree, each leaf must be in black colour

→ same number of black nodes should be present at ~~each~~ all path of tree.



8. B-tree of order 3



9. Split operation in B-tree

① find the median of the full node

② Create a new leaf node and copy into it all the keys which appear at the the median

③ Move up the median to an appropriate position in the parent of this node

④ add an additional child pointer from the parent to the new node

⑤ add the new key at the right location in the child nodes of median.

10. inserting data into red black tree.

① Check if the tree is empty

② if yes, insert new node as root with colour black and exit

③ if no, insert new node as leaf node with color red

④ if parent of new node is black then exit from operation

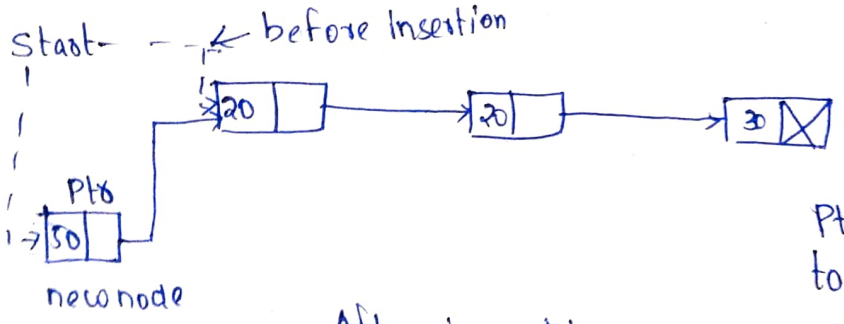
⑤ if parent of new node is red then check colour of parent node's sibling of new node.

⑥ If it is coloured black or null, then make suitable rotation & recolour it

⑦ if it is coloured red then perform recolour until the tree become red black tree.

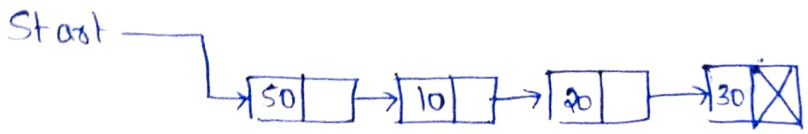
Part - B

11 a. Insertion at beginning



Ptr is the pointer which points to the node that has been inserted.

After insertion

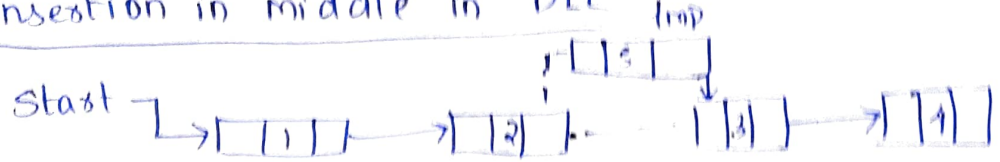


After insertion


```
Struct node * tmp
tmp = malloc (size of (struct node)) ;
tmp -> info = data
tmp -> prev = Null
tmp -> next = start
start <-> prev = tmp
start -> tmp
```

start points to the first node of doubly linked list. for insertion at beginning we assign the value of start to the next part of tmp node and address of tmp node to previous of start.

Insertion in middle in DLL



For inserting the element in middle in DLL we assign the address of inserted node to the prev. part of next node. Then assign the next part of prev. node to then next part of insertednod. Address of prev. node is assigned to prev part of inserted node and address of inserted node will be assigned to next part of previous nod..

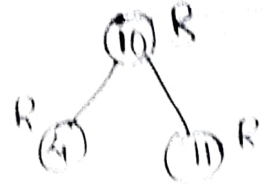
12.
(a) Deletion In Red black Tree.

13a

Deletion in Red black tree

(1) perform similar binary search tree (BST) delete

eg: delete node x from RBT



(2) In delete, the main violated property is changed black heigh in subtrees deletion of a black node may cause reduced black height in one root to leaf

(3) The procedure for deleting a node in RB based on deletion in BST. Here we use a procedure called "TRANSPLANT" subroutine for deleting

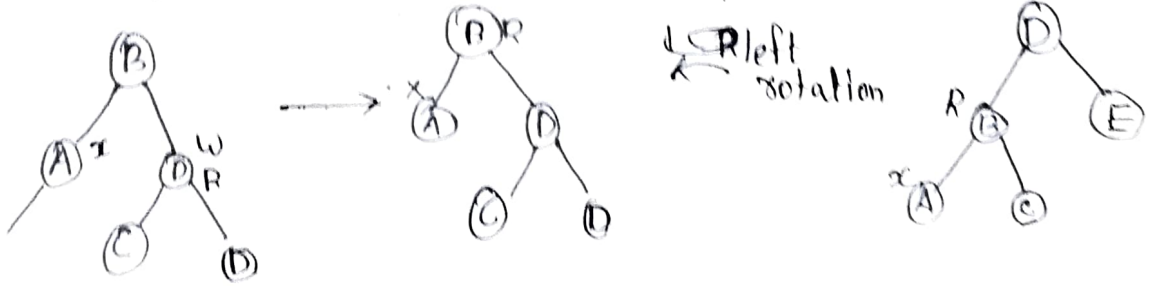
Red black tree delete fixup (T, x)

Color of x = black and x is not root

Case 1:- x's sibling w is red

(1) change color of w and p(x)

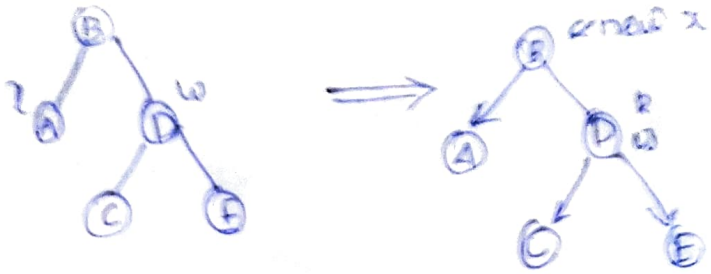
(2) perform left rotation on p(x)



Case 2:- x's sibling w is black and left child is black and right child is black

(1) Change the Colour of w

(2) make p(x) as new x

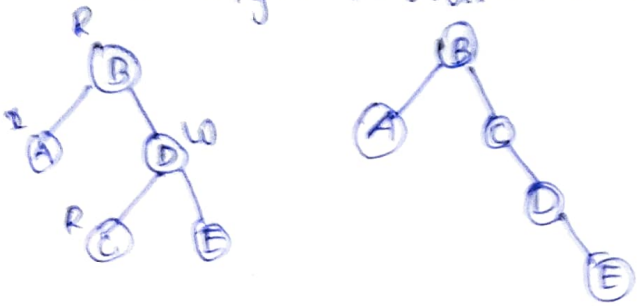


Case 3:- x's sibling w is black and left child is red and w's right child is black

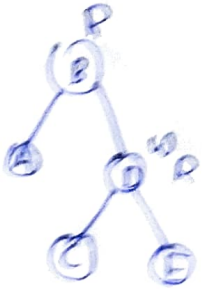
then:-

① change the colors of w & its left child

② Right rotation of w



Right rotation



Case 4:- x's sibling w is black and w's right child is red (Case 2)

1. Change color of w as Colors of p(x)
2. Make color of p(x) to black
3. Change color of right child of w
4. left rotate p(x)
5. make x as root

14. Deletion operation in B tree

There are two case in deletion in Btree

→ deletion from leaf

→ deletion from non leaf

① Deletion from leaf node

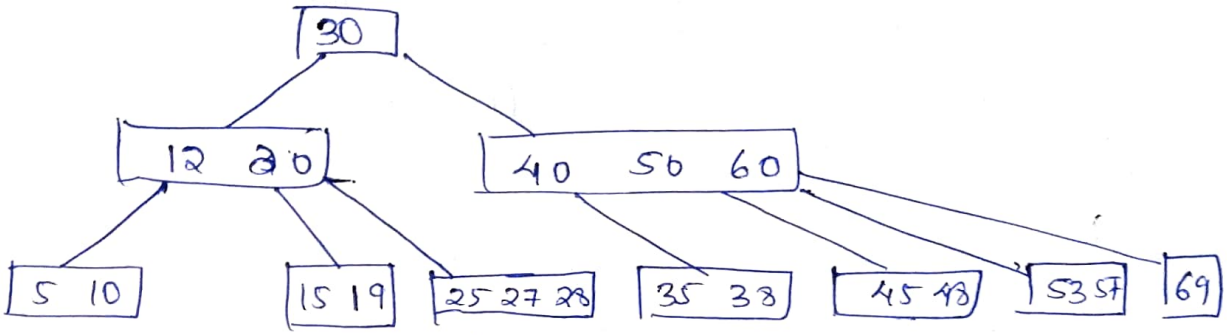
Steps → Check whether the node has minimum keys

→ if yes, then delete the key and Shift other keys of the node

→ If no, check whether the keys in left side is minimum.
→ if yes, then borrow from left sibling
→ if no, check whether the keys in Right side is minimum
→ if yes, borrow from the right sibling
→ if no, then combine the node with left or right sibling

Deletion in non leaf node

Replace the key by its Successor and delelte the successor which will always be in leaf node.



To delete 12, Successor of 12 is 15, so copy 15 to 12. Then delete 15 from leaf by borrowing a key from right sibling and move to parent, so the Seperation key moves down.

12b. Insertion in binary Search tree

Ans. When an element is inserted to BST, first locate its proper location, start searching from the root node, then if the data is less than key value, search the empty location in the L. subtree and insert the data. Otherwise for the empty location in right subtree and insert the data.