11/9/18

Queue:

It is a linear data structure. Follows FIFO. Insertion and deletion from opposite ends.

Deletion:

① Both F and R are -1

Insertion:

① If F=R=-1

② If R= max-1 ⇒ overflow

③ R++ ⇒ insert value

* Insertion and deletion: Array based:

```
#define max 30
int F=R=-1;
int arr[max];
void insertQ(int arr[ ], int num)
{
   if(R==max-1)
  { printf("Overflow");
   return();
   }
   else if(F==-1)
   {
    F=R=0;
    arr[R]=num;
    }
   else
   {
    R=R+1;
    arr[R]=num;
    }
  }
int delQ(int arr[ ])
{
 if(f==-1)
 {
 printf("Empty queue.");
 return();
 }
 else if (F==R)
```

```
{
  item = arr[F];
  F = -1;
  R = -1;
}
else
{
  item = arr[F];
  return(item);
}
return item.
}
```

Linked List based:

```
typedef struct node
{
  int data;
  struct node* next;
}Queue;
Queue * Front = NULL;
```

Circular Queue:

```
void insert(int value)
{
  if (F==R && R==max-1) || (R = F-1)
  {
    printf("Queue is full");
    return;
  }
  else if (F==-1)
  { F=R=0; }
    arr[R] = value;
  }
  else if(R== max-1 && F!=0)
  { R=0;
    arr[R] = value;
  }
  else
  {
    R=R+1;
    arr[R] = value;
  }
}
```

```
~~int C(int front);~~
{
  if (front
int C(int arr[] [illegible]
{
  if arr[illegible] != -1;
```

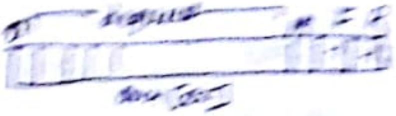
12/9/18

Doubly Ended Queue (Dequeue):

Insertion from both ends, deletion from

Variations:

1. Output restricted (deletion from one end) F
2. Input restricted (insertion from one end) F



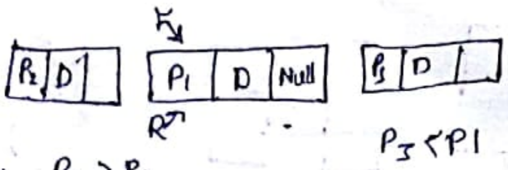
```
typedef struct queue
{
  int arr[SIZE];
  int front, rear;
} dequeue; dequeue q1, q1.front = -1, q1.rear = -1;
void InsertR(dequeue *);
void InsertF(dequeue *);
void DeleteF(dequeue *);
void DeleteR(dequeue *);
int main()
{
  dequeue q1;
  ===
  case 1: InsertR(&q1);
  ==
}
void InsertR(dequeue *p, int item)
{
  ~~if(p->front == (p->rear+1)%SIZE)~~
  if(~~p->front == 0 &&~~ p->rear == SIZE-1)
  {
    printf("Queue is full");
    return;
  }
  else if (p->front == -1)
  {
    p->front = 0;
    p->rear = 0;
    p->arr[p->rear] = item;
  }}
```

```
else
{
  p->rear = p->rear+1;
  p->arr [p->rear] = item;
 }
}
void InsDeQF ( dequeue *p, int item)
{
 if(p->front==0)
 {
  printf(" Queue is full");
  return;
 }
 else if (p->front==-1)
 {
   p->front = p->rear =0;
   p->arr [p->front] = item;
 }
 else
 {
   p->front = p->front-1;
   p->arr [p->front]= item;
  }
 }
void DeldeQR(dequeue *p)
{
   int item;
   if(p->front==-1)
   {
    printf(" underflow");
    return;
   }
  else if(p->front == p->rear)
  {
    item = p->arr[p->front];
    p->front = p->rear = -1;
  }
  else
  {
    item= p->arr[p->rear];
    p->rear = p->rear-1;
  }
  return item;
}
```

Scanned by CamScanner

Priority Queue:

If the new node has priority higher than current, it is inserted to the left of the node, otherwise to the right, $P_2 > P_1$



- Insertion can be done sorted or unsorted.

For deletion, we need to find out the highest priority node each time. That element is deleted first.

13/9/18

More Examples on Recursion:

* Draw n concentric circles.

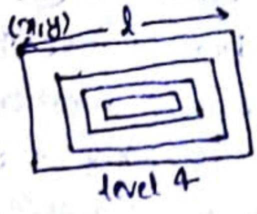
```
DrawCircle (int x, int y, int r, int n)
{
  if(n==0)
   return;
  else
   {
     DrawCircle(x,y,r-10,n-1);
     Circle (x,y,r);
   }
}
```

To draw the bigger circle first, call the in-built function circle before recursion of DrawCircle().

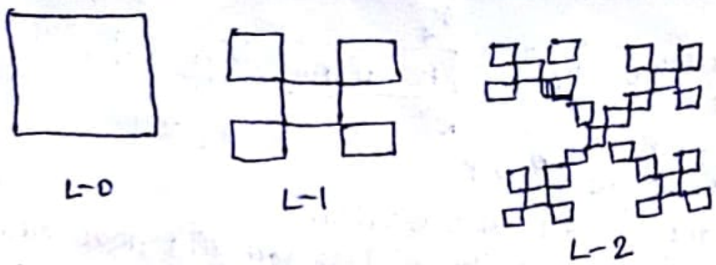
- Without using in-built function:

```
main()
{
  int gd = DETECT, gm;
  initgraph(&gd, &gm, "C:\\");
```

* Draw n ~~rectangles~~ squares:



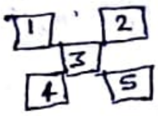

```
Draw~~Rectangle~~Square (int x, int y, int l, int n)
{
  if(n==0)
   return;
  else
   {
     Draw~~Rectangle~~Square (x+l/4, y+l/4, l/2, n-1);
square ~~rectangle~~(x,y,l);
   }
}
```



* Complex Geometrical Shapes:

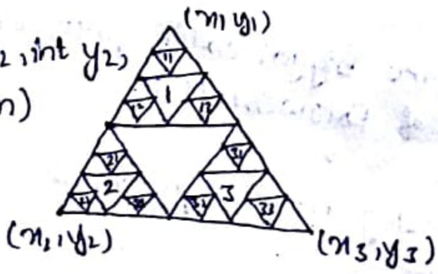
- DrawFractal(int x, int y, int l, int n)

```
{
  if(n==0)
  Square(x,y,l);  Delay(100);
  else
  {
   DrawFractal(x,y,l/3,n-1);  1
   DrawFractal(x+2*l/3,y,l/3,n-1);  2
   DrawFractal(x+l/3,y+l/3,l/3,n-1);  3
   DrawFractal(x,y+2*l/3,l/3,n-1);  4
   DrawFractal(x+2*l/3,y+2*l/3,l/3,n-1);  5
  }
}
```



- Sierpinsky Triangle:

```
DrawTriangle(int x1,int y1,int x2,int y2,
             int x3,int y3,int n)
{
 if(n==0)
  return;
 else
  {
   DrawTriangle(x1,y1,(x1+x2)/2,(y1+y2)/2,(x1+x3)/2,(y1+y3)/2,n-1);
   DrawTriangle((x1+x2)/2,(y1+y2)/2,x2,y2,(x2+x3)/2,(y2+y3)/2,n-1);
   DrawTriangle((x1+x2)/2,(y1+y2)/2,(x2+x3)/2,(y2+y3)/2,x3,y3,n-1);
   line(x1,y1,x2,y2);
   line(x2,y2,x3,y3);
   line(x1,y1,x3,y3);
  }
}
```



Koch Curve:

18/9/2018

```
KoshCurve(int x1, int y1, int x2, int y2, int n)
{
    int x3, y3, x4, y4, x, y;
    if(n == 1)
    Line(x1, y1, x2, y2);
    else
    {
        x3 = (2x1 + x2)/3;
        y3 = (2y1 + y2)/3;
        x4 = (x1 + 2x2)/3;
        y4 = (y1 + 2y2)/3;
        x =
        y =
        KoshCurve(x1, y1, x3, y3, n-1);
        KoshCurve(x3, y3, x, y, n-1);
        KoshCurve(x, y, x4, y4, n-1);
        KoshCurve(x4, y4, x2, y2, n-1);
```

* Backtracking:

n Queen Problem:

n queens are to be placed on a nxn board such that no 2 queens are in a vertically, horizontally or diagonally in the same line

8x8

Q_1	x		x				
	x	x	Q_4'	Q_5			
	Q_2						
				~~Q_5~~	~~Q_6~~		
		Q_3					
			ii				
			iii	(i)			

No place is left for Q_6, so now start backtracking. Move Q_5 to new position, still no place found, bt again and place Q_4 to new position. Move Q_4 to position (ii) Now place Q_5

Move Q_4 to position (iii). Now Q_5 has two possible positions. Place it on (i). Now Q_6 has one possible position and so does Q_7. But now Q_8 can not be placed. Backtrack again

4x4

~~Q~~		Q_3	
Q_1			
	~~Q_2~~		Q_4
	Q_2		

* Rat in a maze Problem:

R							
	X		X		X		X
X		X		X			
				X		X	X
		X	X		X	X	X
					X	X	X
X					X	X	X
X							F

The rat has to reach its destination, i.e, the food using four movements only - up, down, right and left. There are certain blockages in the path.

19/8/18

Insertion Sort:

If $A[\,] = \{a_1, a_2, a_3 \ldots\ldots a_n\}$

1. Sort the first element.
2. Compare a_2 and a_1, swap if $a_2 < a_1$. {for ascending order}
3. Compare a_3 and a_2. If $a_3 < a_2$ compare a_3 with a_1. If $a_3 < a_1$
4. Store a_3 in a variable and shift both to right
5. $a_4 < a_3 \Rightarrow$ compare with a_2

eg. 5, 10, 2, 1, 15, 20, 4

5 10 2 1 15 20 4

2 5 10 1 15 20 4

1 2 5 10 15 20 4

1 2 5 10 15 20 4

1 2 5 10 15 20 4

1 2 4 5 10 15 20

No. of comparison = 13

* Best case: already sorted, No. of comparisons = no. of elements = n
* Worst case: sorted in reverse order, No. of comparisons = n^2

Quick Sort:

Divide and conquer algorithm

$A[\,] = \{a_1, a_2, a_3, a_4 \ldots\ldots a_n\}$

Consider an element as pivot and find its right place. The elements on the right are bigger and on the left are smaller but are not sorted among themselves.

a_1 is pivot. If $a_2 < a_1$, swap. Start from right. Compare a_n with a_{n-1}, $a_{n-2} \ldots a_1$

Compare a_n and a_1. If suppose we find $a_{n-1} < a_n \Rightarrow$ swap. Else continue comparison of a_n with $a_{n-2}, a_{n-3} \ldots a_1$.

$a_1 \; a_2 \; a_3 \ldots\ldots a_n$

$a_{n-1} \; a_2 \; a_3 \ldots\ldots a_1 \; a_n$

* Worst case: already sorted array
  No. of comparisons = n^2
  {All elements have to be compared.}
* Best case: If pivot divides the list in two equal halves.
  Time complexity = $n \log n$

eg. 2 10 5 15 1 6 4 20 12

~~2 5 15 6 4 20~~

1 2 10 5 15 1 6 4 20 12

1 2 5 15 10 6 4 20 12

1 2 4 15 10 6 5 20 12

1 2 4 5 10 6 15 20 12

$a_1 \; a_2 \; a_3 \ldots\ldots a_n$

$n/2 \quad a_1 \quad n/2$

$TC = n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots \; n(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots)$

$2n$ [illegible]

3/10/2018

Tree:

It is a non-linear data structure used to represent data in hierarchichal order.

Root: Every tree has a unique root.

Note: The data elements in a tree are represented by nodes.

Edges: Lines which connect two nodes.

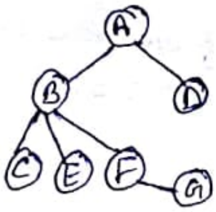
Parent: If there are two nodes connected with an edge, the (source) predecessor node of connection is called parent.

* Except root, all nodes must have a parent.

Child: The immediate successor node of any node is called child.

Degree: Degree of a node is the maximum no. of children.

Degree of tree is the max degree of node having no. of children. ~~of any node~~



Level: Level of root is 0. Its child has level 1 and so on.

Siblings: All children of a node are siblings. eg B & D, C, E, F

Height of a Tree: Max no. of edges in the longest path starting from the root.

Degree of tree = 3   Height of given tree = 3
Height of D = 1

Path: Combination of edges from root to a particular node.

Depth of a Tree: No. of edges from the leaf. eg Depth of this tree = 3
Depth of B = 2, Depth of D = 0.

Leaf: The nodes which do not have any child. eg D, E, G, C
or
External nodes or Terminal nodes

Internal nodes: Nodes which have at least one child. eg A, B, F.

Subtree: Nodes attached to left/right child constitute left/right subtree.

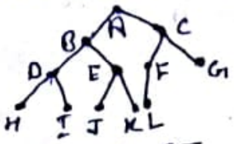
Binary Tree:

A tree having max two children at any node.

Full Binary Tree: A BT in which all levels have max possible no. of nodes.
* No. of nodes possible at each level = 2^l

~~Complete Binary Tree~~: No. of internal nodes = n-1   {n = no. of leaves}
No. of external nodes = e+1   {e = no. of internal nodes}

Complete Binary Tree: It is a subset of full binary tree.
All levels have max ~~no~~ possible no. of nodes except the last level and in the last level, nodes are ~~aligned~~ as much left aligned as possible.



Comple BT

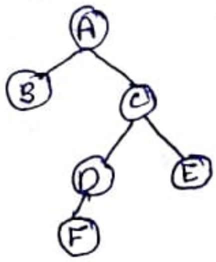
2 Tree: A BT in which each node has either two or zero child.

Representation of Binary Tree:

1. Sequential (array based) representation
2. Linked list representation

Sequential Representation:

Declare a 1D array. Assign first place to root. If there is a node stored at ith location, then its left child is stored at (2*i)th location and right child at (2*i+1)th location.



1	2	3	4	5	6	7	8	9	10	11	12
A	B	C			D	E					F

If there is a node at ith location, its parent should be at $\boxed{[i/2]}$th location.

Greatest Integer

If $i=6$, parent is at $[6/2]=3$

If $i=7$, parent is at $[7/2]=3$

Drawbacks:

i.) Wastage of memory

ii.) Insertion, deletion is difficult.

Linked List Representation:

ptr to left child	data	ptr to right child

...

Tree Traversal:

There are three methods of traversing a tree:

i.) In-order

ii.) Pre-order

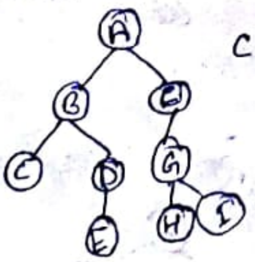
iii.) Post-order

i.) In order Traversal:

Steps to traverse a tree ~~by any method~~:

1. Traverse the left subtree.
2. Visit root.
3. Traverse right subtree.

Start traversing from left subtree. Print the node encountered second time.

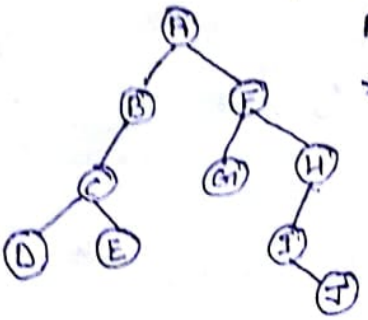


C B E D A H G I F

4/10/2018

(ii) Pre-order Traversal:

1. Process the root.
2. Traverse left subtree in pre-order
3. Traverse right subtree in preorder.



A B C D E F G H I J

* Follow the path and print the node first encountered.

(iii) Post-order Traversal:

1. Traverse left subtree in postorder.
2. Traverse right subtree in post-order.
3. Process the root.

In the above binary tree, post-order traversal gives following results

DECBGJIHFA

* The node is printed when you do not get a chance to revisit that node ~~again~~. (in the last visit of the node)

In-order: DCEBAGFIJH

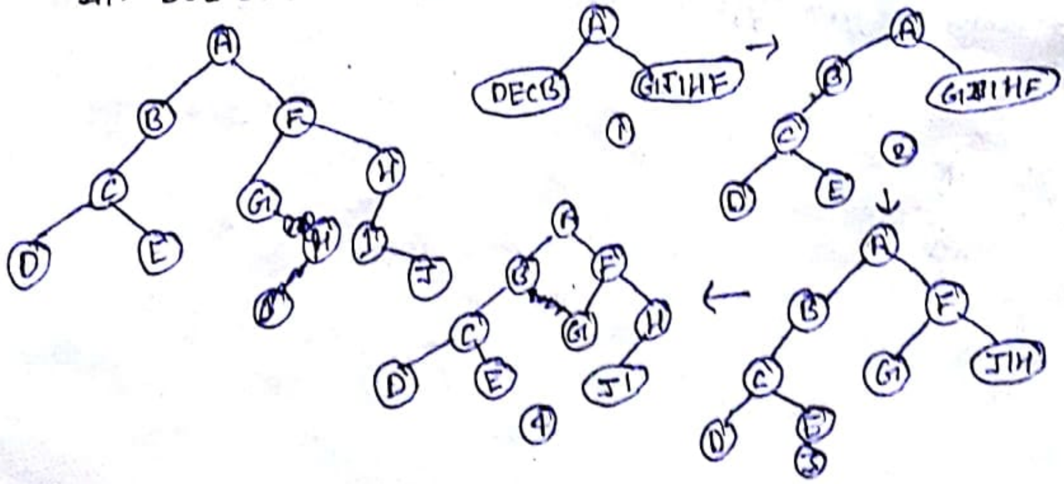
Construction of a Tree using two given traversal orders:

* A unique tree can be drawn provided either of following is known:
  i) In-order and pre-order
  ii) In-order and post-order

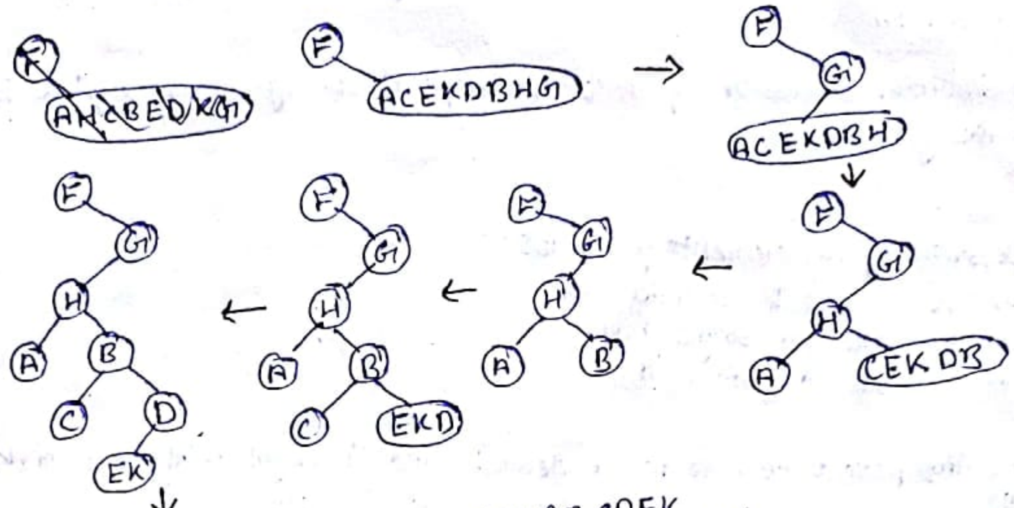
* The first node of pre-order and last node of post-order gives root.
* In-order separates the left subtree from the right one.
* Pre-order of ~~left~~ subtree separates the right and left children.

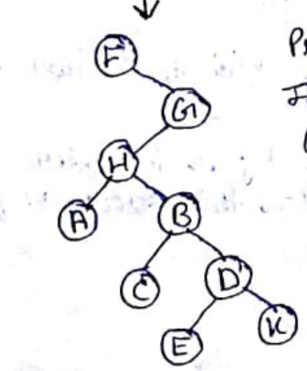
eg. Pre - ABCDEFGHIJ
In - DCEBAGFIJH

Post - DECBGJIHFA
In - DCEBAGFIJH



eg. In- F A H C B E D K G , Post- A C E K D B H G F





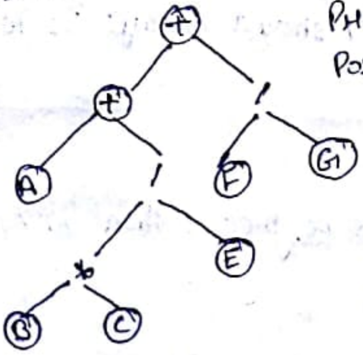
Pre-order - FGHABCDEK

If we are given pre and post-order of this tree (or any tree), it is impossible to draw a unique tree using them.

** We can draw a binary tree for a given expression (which does not include unary operators).

Operators - internal nodes, operands - leaf nodes.

eg. A+B*C/E+F/G



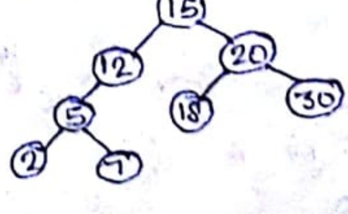
Prefix : ++A/*BCE/FG

Postfix : ABC*E/+FG/+

Binary Search Tree (BST):

For any node, left subtree always contains smaller nodes and the right subtree contains equal or greater nodes.

eg. 15 12 5 2 7 20 30 18



Make first element root. 12 is <, so left. 5<15&12 => left

2<15,12,5 => left

7<15,12 but >5 => right to 5

20>15 => right

30>15,20 => right

18 < 30,20 => left to 20

* If BST is balanced, we can perform binary search in time complexity of order n.

* The in-order successor of node can replace it. eg. 18 can replace 15 in the given tree.

10/10/19

Deletion of an element from BST:

i) The node without child
ii) The node has single child
iii) The node has two child

i.) Find the parent of node to be deleted. Find its right child and make it NULL.

ii) The parent's address field is replaced by the child of the node to be deleted.

iii) In this case the node to be deleted is replaced by its in-order successor, Then check where in which category does this successor fall b/w i) and ii) cases.

Heap Tree:

It is of two types:

i) Min heap
ii) Max heap

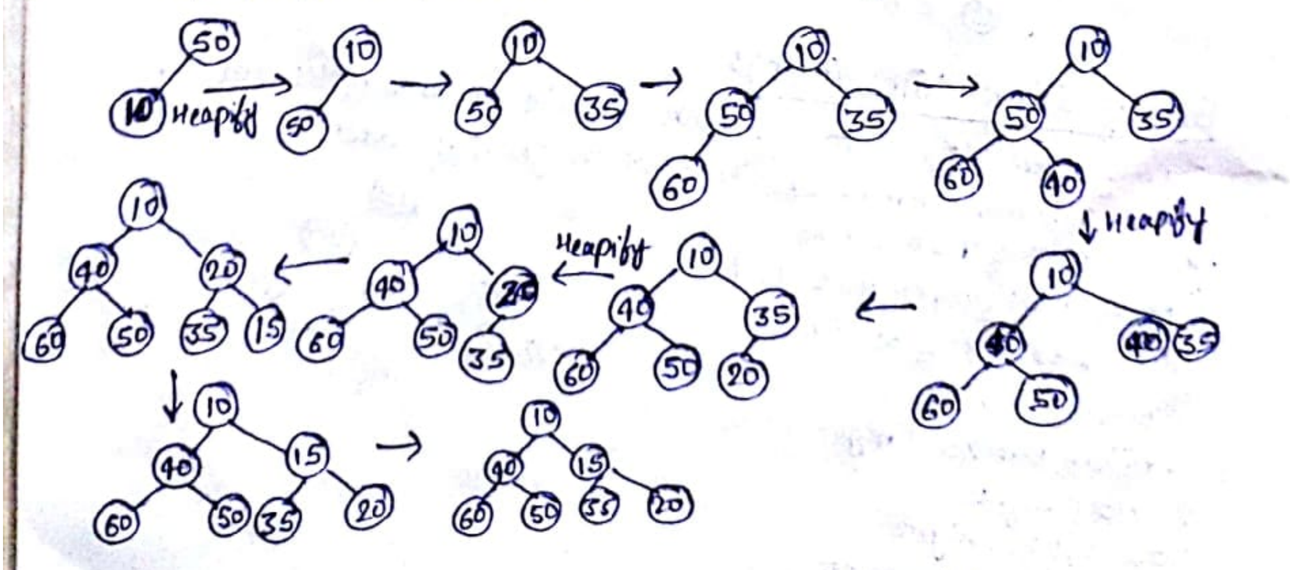
Min Heap Tree:

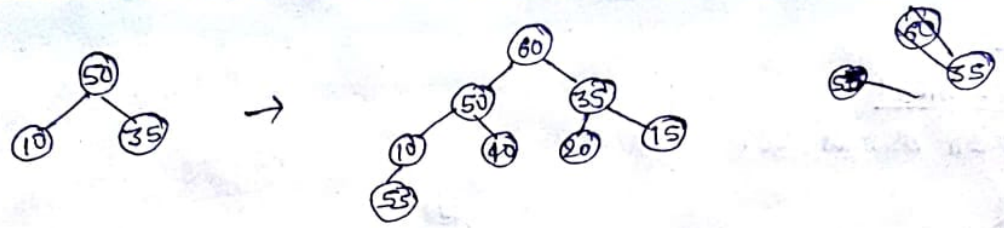
i.) It is a complete binary tree.
ii.) The parent node (including root) should always be less than or equal to its children.

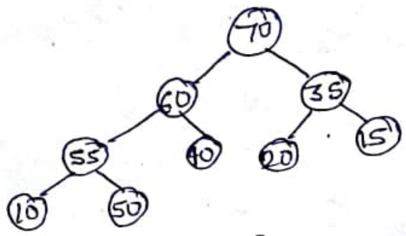
Max Heap Tree:

* It is a complete binary tree.
* The parent node should always be greater than or equal to its children.

50, 10, 35, 60, 40, 20, 15, 55, 70

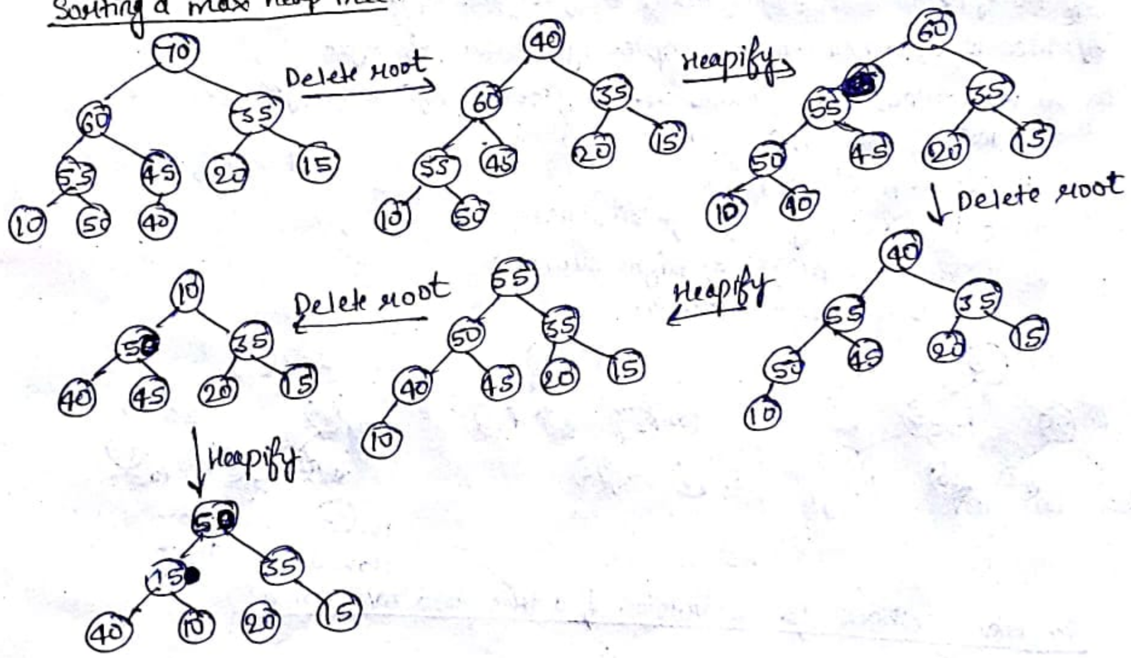






Max Heap

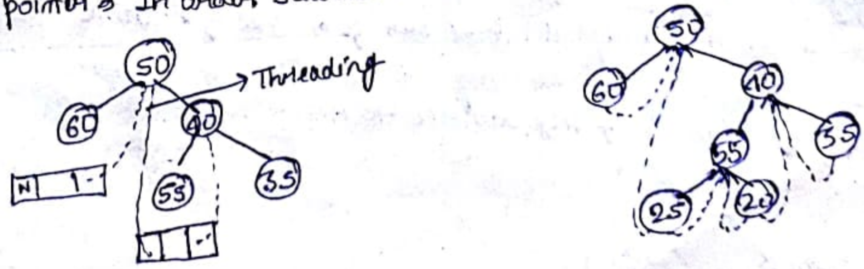
Sorting a max heap tree:



Threaded Binary Tree:

It is a 2 tree or extended tree

Left pointer ⇒ In order predecessor
Right pointer ⇒ In order successor



11/10/2018

AVL Tree:

It is a balanced BST

Balanced BST (skewed) Skewed to the left

If no. of elements = n, Time taken is of order of n.

If tree is balanced, time complexity = order of $\log_2 n$

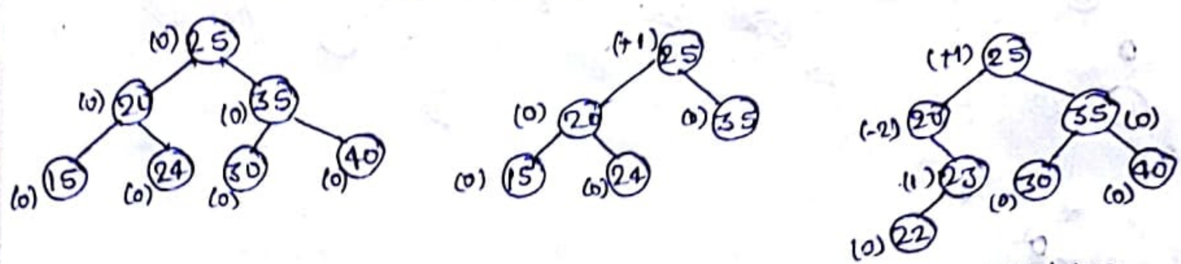
* In an AVL tree, the difference of heights of left and right subtree should always be +1, 0 or -1.

$BF = h(T^L) - h(T^R) \leq 1$

where $h(T^L)$ = height of left subtree

$h(T^R)$ = height of right subtree

BF = Balance Factor



Not an AVL tree

Rotation methods for conversion of a tree into AVL tree:

Find a node A such that it is the first ancestor of the node that we insert to make the tree non-AVL.

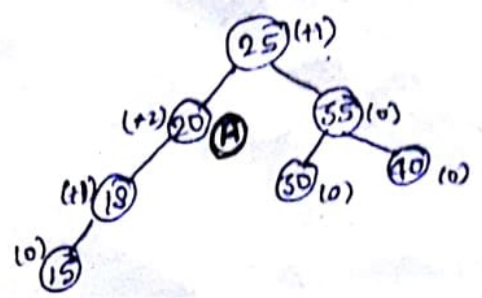
Start from inserted node towards root, find node that does not have its BF as 0, 1 or -1.

Rotate the non-AVL tree around the ancestor.

Based on the ancestor, there are four kinds of rotations.

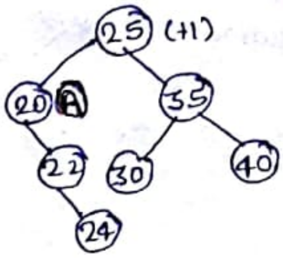
i) LL

LL = Left subtree of left subtree to A.



ii:) RR

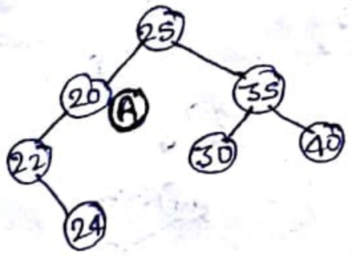
RR = Rotate along right subtree of right subtree from A.



iii:) LR

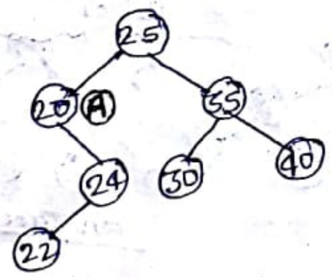
LR = Rotate along right subtree of left subtree from A.

Parent of inserted node becomes root, and the inserted node replaces its parent.

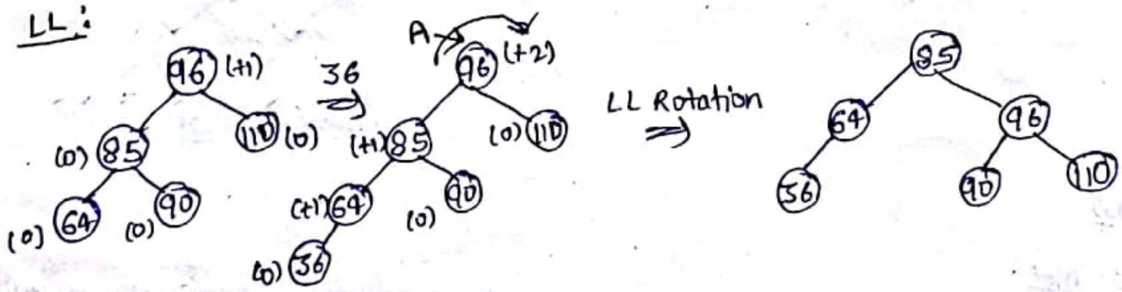


iv) RL

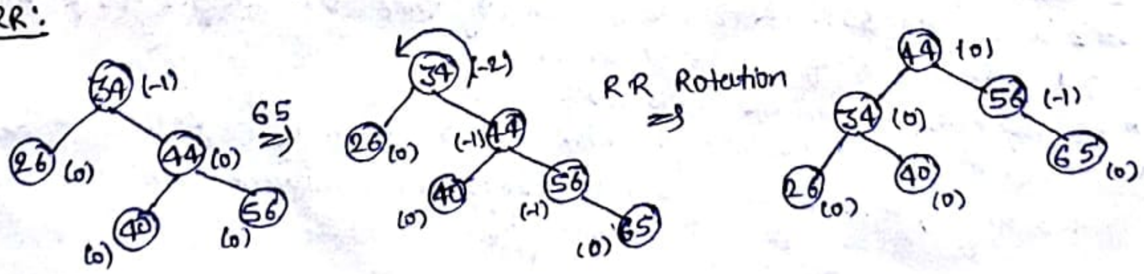
RL = Rotate along left subtree of right subtree from A.



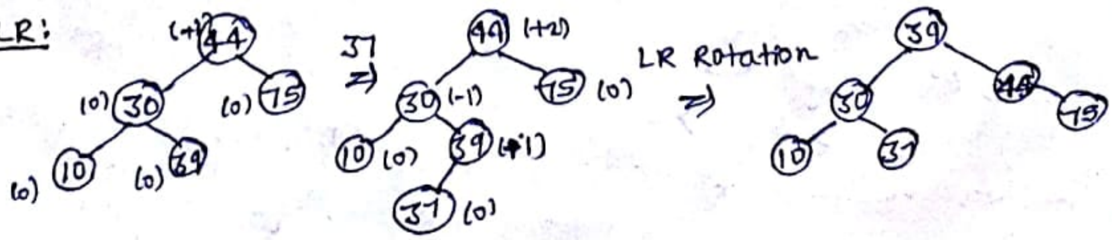
LL:



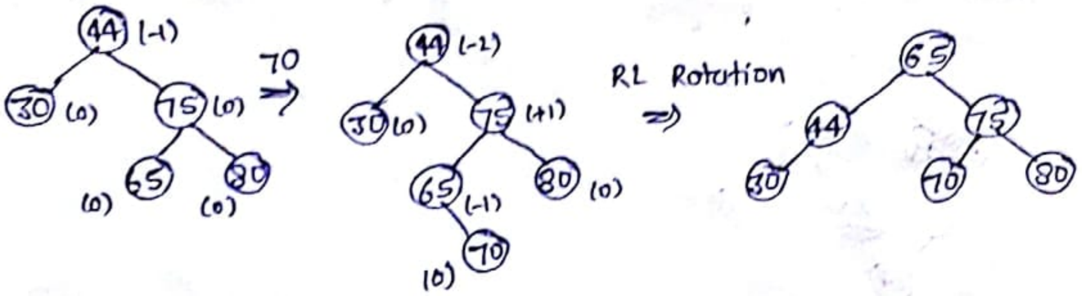
RR:



LR:

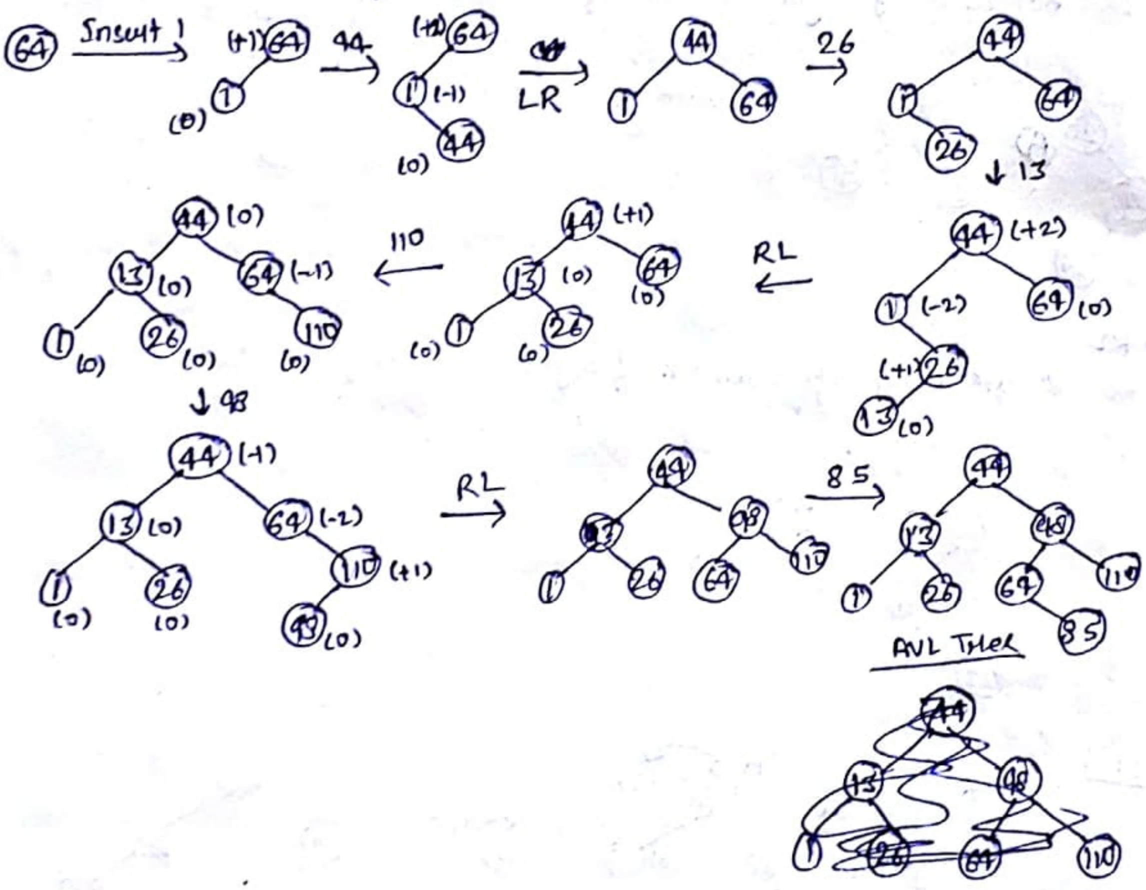


RL:



Construction of an AVL Tree:

Construct an AVL tree from 64, 1, 44, 26, 13, 110, 98, 85



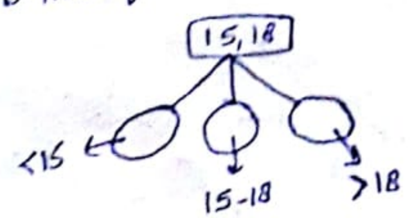
22/10/2018

B-Tree:

It's a balanced BST. A b-tree is an m-way tree that satisfies the properties of a BST. (m = no. of possible children)

* All leaf nodes are on the same level.
* Root can have minimum two and maximum m children (m-1 keys / elements)

eg. A B-tree of order 3 can have max 2 keys and 3 children (for root node) child)

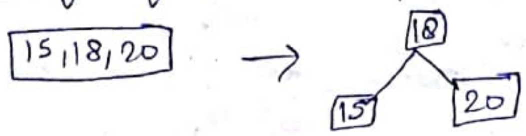


~~Root can have minimum~~ * Non-Root node $\lceil \frac{m}{2} \rceil$ childs ($\lceil m/2 \rceil - 1$) ← at least

at most m child, (m-1) keys

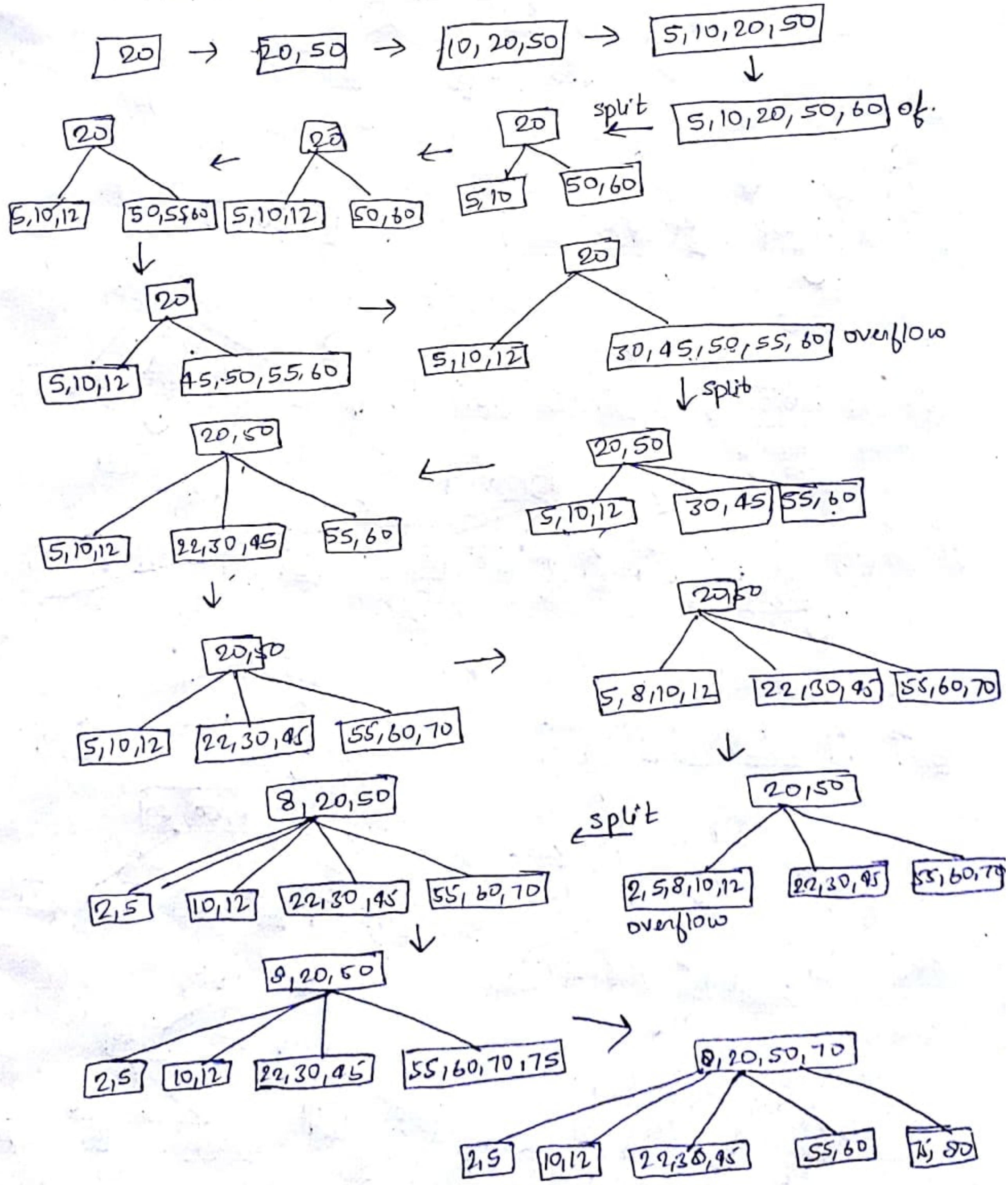
Scanned by CamScanner

In case of overflow, the node splits. eg if m=3,

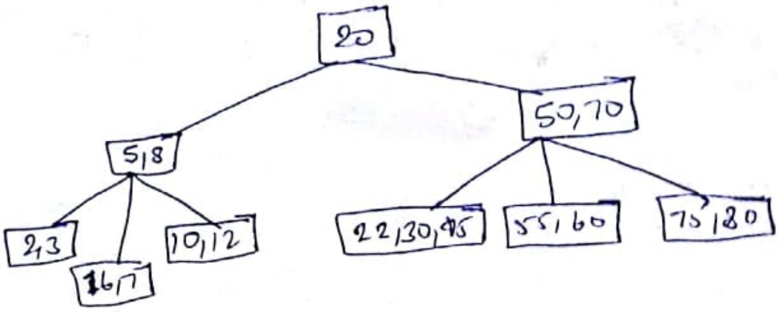


m=4,

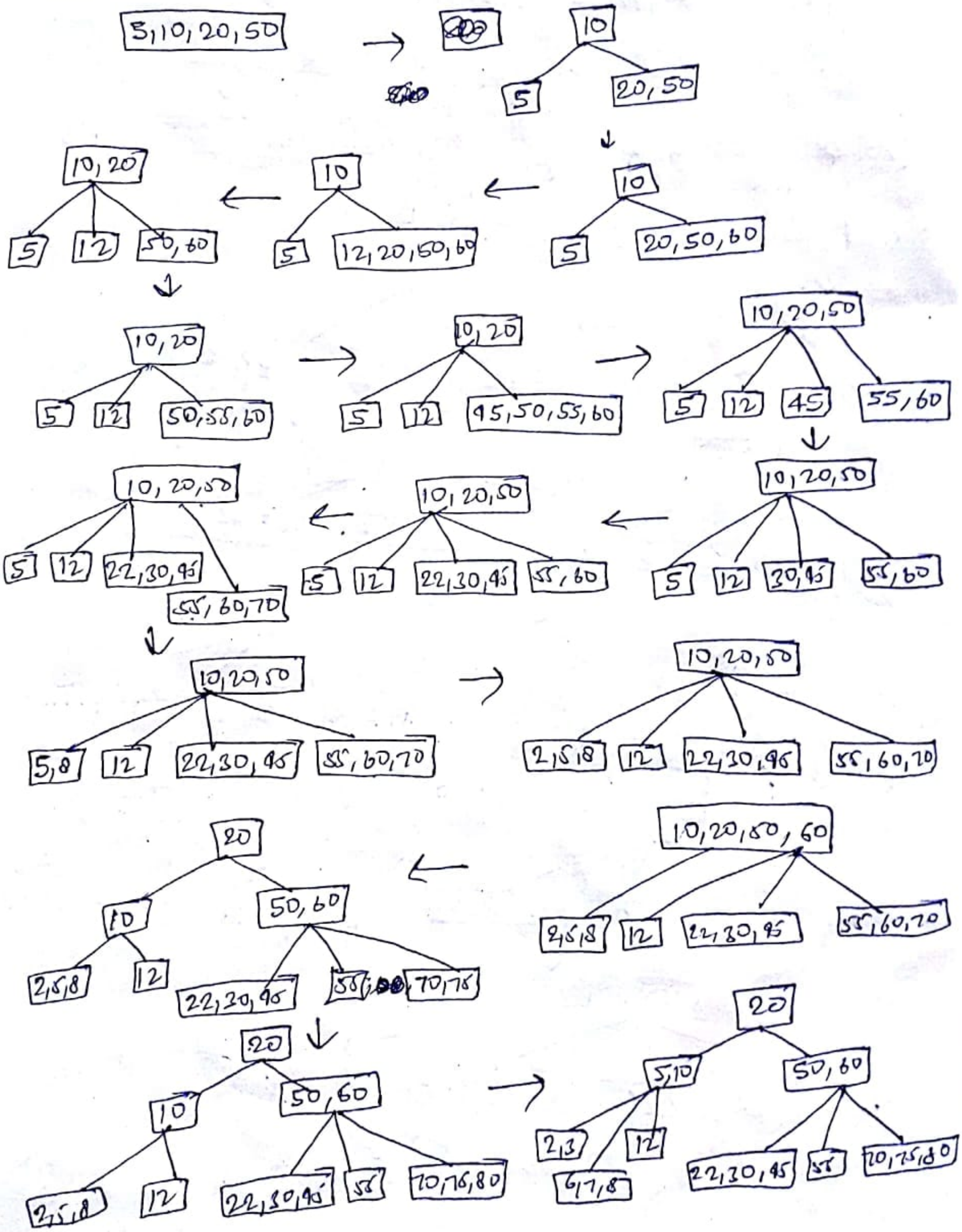
eg. Create a B-tree of order 5 with elements:
20, 50, 10, 5, 60, 12, 55, 45, 30, 22, 70, 8, 2, 75, 80



Now add 6,3,7



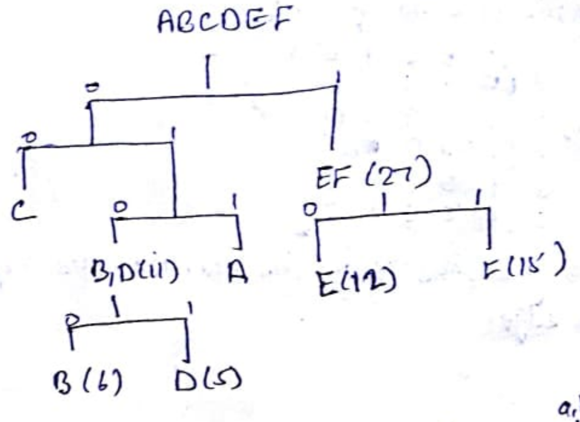
B-tree of order 4:



Huffman Tree:

Arrange in decreasing order of frequency.
Form the least frequencies in a B-tree.
Delete them from list and combine them.
Again, form pair of least frequencies,
combine and place in correct order.

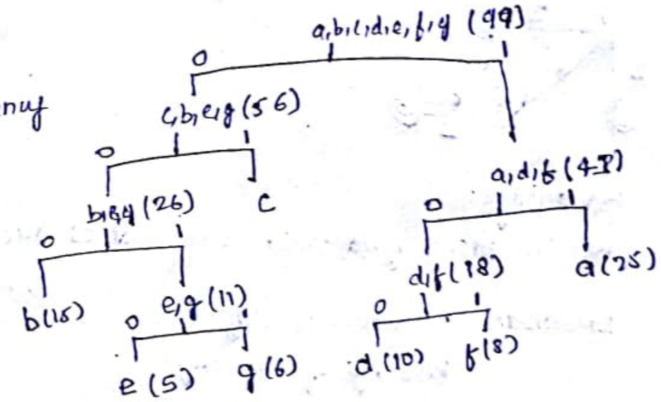
Symbol	Frequency
A	10
B	5
C	20
D	6
E	12
F	15



A = 011
B = 0100
C = 00
D = 0101
E = 10
F = 11

25/10/2018

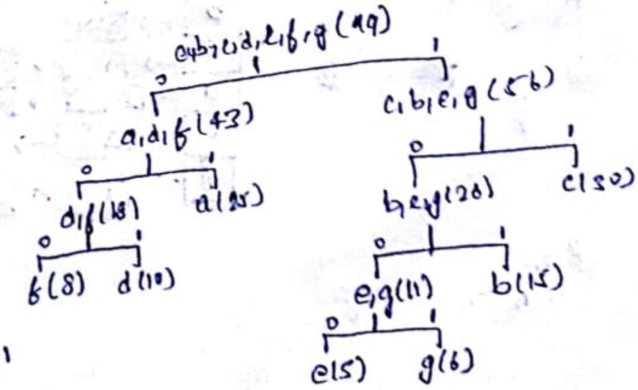
		Symbol/Data	Frequency
c	30	a	25
a	25	b	15
b	15	c	30
d	10	d	10
f	8	e	5
g	6	f	8
e	5	g	6



c	30	c	30	c	30	a,d,f	43	c,b,e,g	56	a,b,c,d,e,f,g (99)	
a	25	a	25	b,e,g	26	c	30	a,d,f	43		
b	15	d,f	18	a	25	b,e,g	26				
d	10	b	15	d,f	18						
f	8	e,g	11								
e,g	11										

Draw step by step

a – 01
b – 10
c – 011
d – 1001
e – 1000
f – 000
g – 1001



Scanned by CamScanner

30/10/2018

Graph:

Non-linear data structure, can be directed or undirected

Representation of Graph:

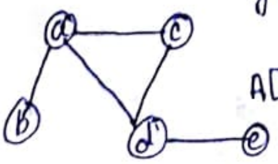
i.) Adjacency matrix

ii.) Adjacency list

Adjacency Matrix: (Undirected graph)

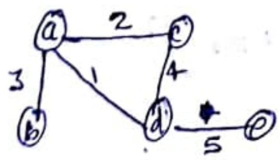
Consider a $n \times n$ matrix, $n =$ no. of vertices

eg $n = 5$



$$A[5][5] = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

* If the graph is weighted, the 1's in the matrix are replaced by the weight of the edge.



Advantage:

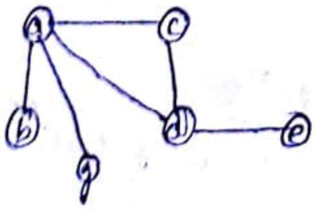
We can easily find which two vertices are connected

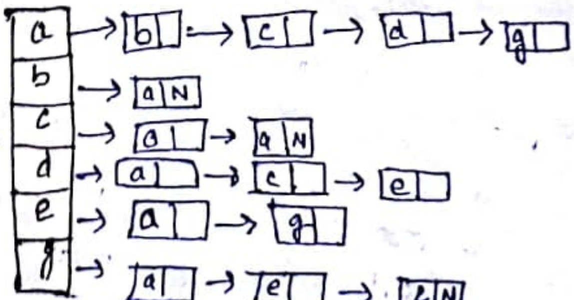
Drawback: Insertion is difficult.

Wastage of memory.

Adjacency List:

Make an array of structures. Form singly linked list to show connections.





Pros:

Insertion is easy.

Cons:

Not easy to find the existence of an edge b/w two vertices.

Applications of graph:

i.) To find the optimum length of wire required to connect computers.

ii.) Find route b/w two places.

Graph Traversal:

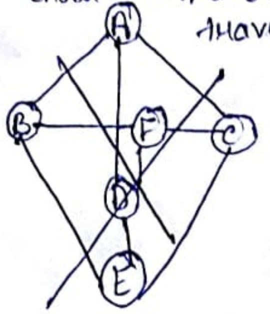
Methods of graph traversal:

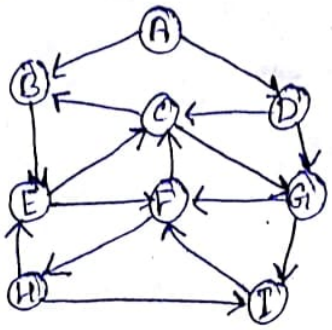
i.) Breadth First Search (BFS)

ii.) Depth First Search (DFS)

BFS:

Choose source. Traverse the neighbours first, then their neighbours. One path is traversed only once.





Q. empty   initially 0s

Visited [ ] = | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Q	Visited [ ]
A	1
ABCD	1111
ABCDE	11111
ABCDEG	1111101
ABCDEGF	1111111
ABCDEGFI	111111101
ABCDEGFIH	111111111

~~Start from A.~~

Print : ABCDEGFIH

DFS:

Stack	Visited [ ]	Pop	Print
E	1	A	A
H	1111	D	AD
I	1111001	G	ADG
F	111101101	I	ADGI
G	111101111	F	ADGIF
D	111111111	H	ADGIFH
C		E	ADGIFHE
B		C	ADGIFHEC
A		B	ADGIFHECB