CHAPTER 1
ARRAYS

One dimensional array

A: array [lb ... ub] of elements

lb : lower boundary
ub : upper boundary
L_0 : Starting location
C : element size
A(i) : i^{th} element size, count

$$Loc.of\ A(i) = L_0 + (i - lb) * C$$

two dimensional Arry

A: array $[b_1 \cdots u_1, b_2 \cdots u_2]$ of elements

rows i , columns j

$r_1 \times r_2$

$(u_1 - b_1 + 1) \quad (u_2 - b_2 + 1)$

RMO (Row major order)

$$L_0 + (i-1)\, r_2$$

↑ In simple notation

In scientific notation:

$$Loc\,A[i,j] = L_0 + [(i - b_1)(u_2 - b_2 + 1) + (j - b_2)] * C$$

CMO: (Colum major order)

$$L_0 + (j-1)\, r_1$$

↑ In simple notation

In scientific notation

$$Loc\,A[i,j] = L_0 + (j - b_2)(u_1 - b_1 + 1) + (i - b_1)] * C$$

Chapter 2
Stack and Queue

Euclid Algorithm

```
GCD(x, y):
    If (y > x)
        GCD(y, x)
    If (y == 0)
        return x
    Otherwise:
        GCD(y, x mod y)
```

GCD: Greatest common division.

* Combination of an objects taken k at a time (Algorithm)

Combination

```
C(n, k):
    If (k == 0 || n == k):
        return 1
    If n > k > 0
    return C(n-1, k) + C(n-1, k-1)
```

Ackerman function:
(not all computable function is recursive)

$$A(x,y) = \begin{cases} y+1 & \text{If } x = 0 \\ A(x-1,1) & \text{If } y = 0 \\ A(x-1, A(x, y-1)) & \end{cases}$$

proper

$A(x-1, A(x, y-1))$ otherwise

Formulae:

$A(0, y) = y + 1$

$A(1, y) = y + 2$

$A(2, y) = 2y + 3$

$A(3, y) = 2^{y+3} - 3$

↑ Important Formula

The tower of Hanoi

Algorithms

(Alway Create your own hypothesis)

Recursive Solution

1. Move n-1 disk from source to Helper
2. move one disk from source to destination
3. Move n-1 disk from Helper to destination.

$((2+3)*4]+[5*((6+7)*8)+9$

Prefix

$+*+234*5678+9$

$++*+234**5+6789$

Prefix Preferred

Rough

$2(5)+3=$

$10+3=13$

$(2,5)$

$2(3)+3=9$

$=4$

$=2^{4+3}-3$

$=2^{3}-3$

$=8-3$

$=5$

4 steps

According to

Invocation

(n-1) ~~S to~~ Source to Auxiliary

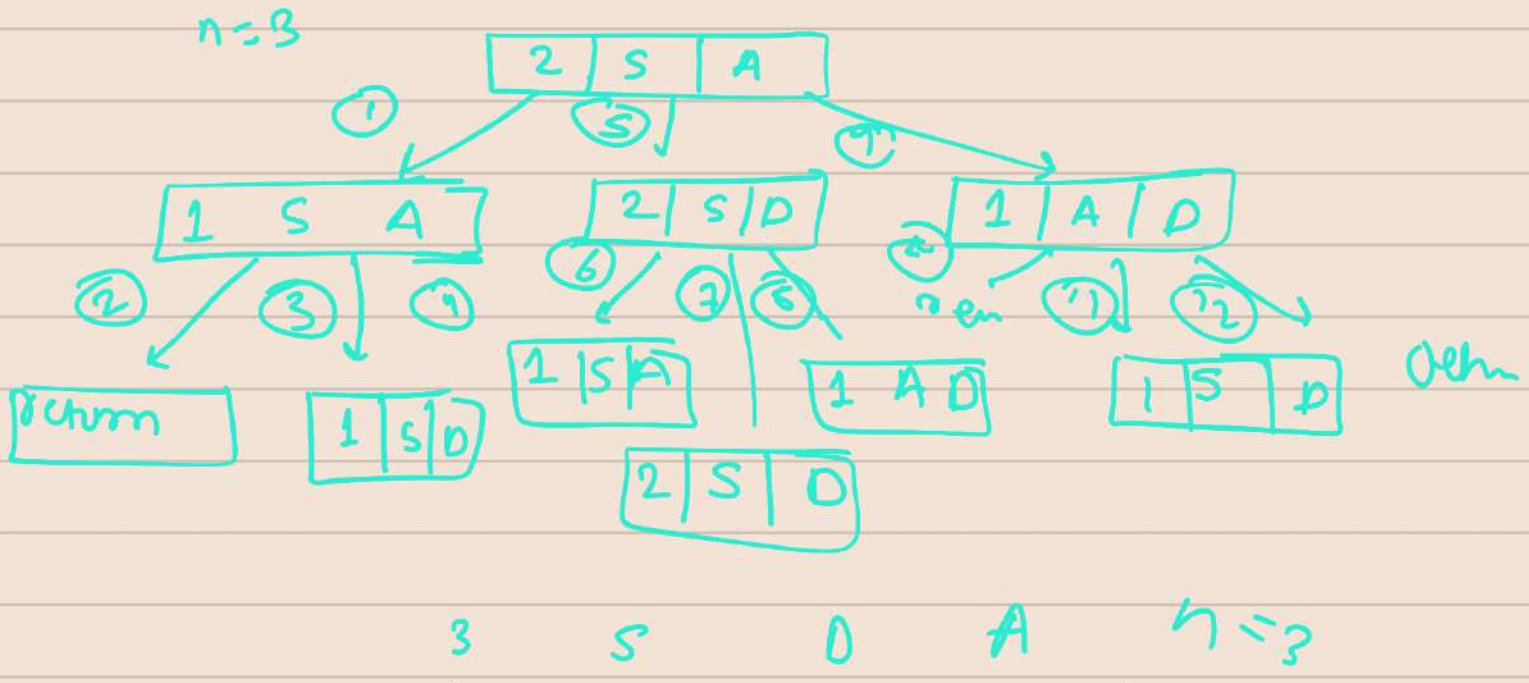
print(n Source to destination)

n-1 Auxiliary to destination

(n-1) Source to Auxiliary

n Source to destination

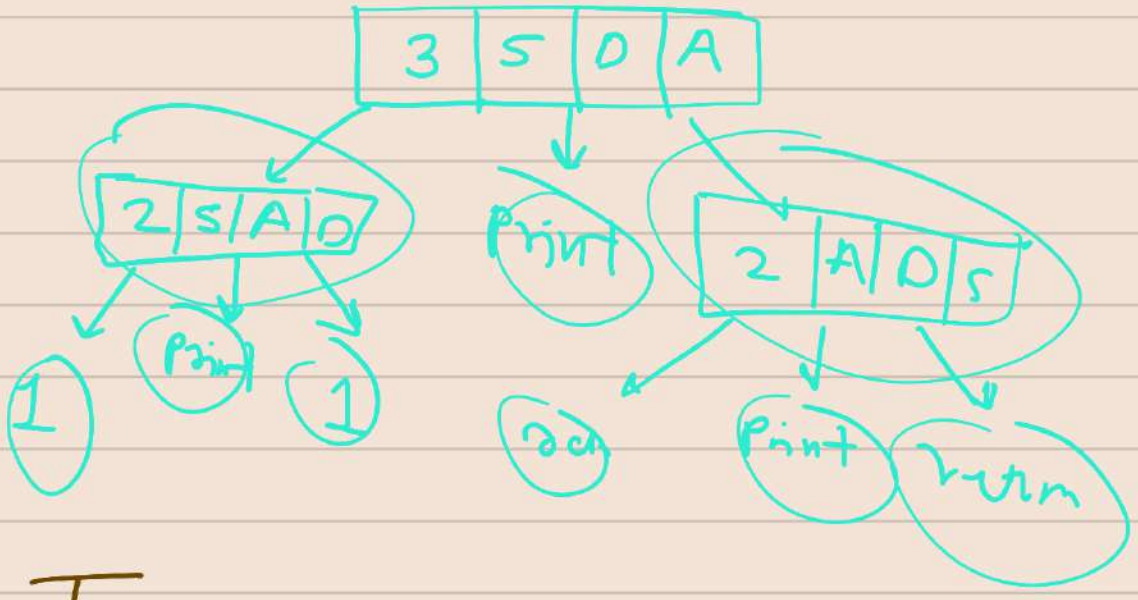
(n-1) Auxiliary to destination



3 S D A n=3

```
def TowerOfHanoi(n , from_rod, to_rod, aux_rod):
    if n == 1:
        print("Move disk 1 from rod",from_rod,"to rod",to_rod)
        return
    TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
    print("Move disk",n,"from rod",from_rod,"to rod",to_rod)
    TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)

# Driver code
n = 4
TowerOfHanoi(n, 'A', 'C', 'B')
# A, C, B are the name of rods
```



Trees

* indegree, outdegree
* root node has degree zero
  rest all has degree one.

* A leaf is node with out degree zero
* Internal node is a node that is neither root nor leaf.
* Height of tree = level of leaf from longest path to root + 1
* Height of empty tree is -1

Binary Formula

Minimum and Maximum Height of tree can related to the number of nodes

$$H_{min} = \lfloor \log_2 N \rfloor + 1 \qquad H_{max} = N$$

- If Height is given the Maximum number of nodes or Minimum number of nodes can be Calculated

$$N_{min} = H \qquad N_{max} = 2^H - 1$$

- Balance factor (Diff between left and right node)

$$B = H_L - H_K$$

Properties: (Mostly Binary tree)

- If depth of tree is given we can find the height of tree:

$2^{h+1} - 1$ where h is a depth of the tree.

↖ This is for binary tree

- h = height / depth

The number of node n in a binary tree of h is at least $n = h+1$ and at-most $n = 2^{h+1} - 1$ where h is a depth of tree.

- The no. of leave nodes L can be found using $L = 2^h$ h = depth of tree / height

- The number of nodes n in a full binary tree can also be found using $n = 2L - 1$ L = Leaf nodes

- The number of null links (i.e absent children of node) in a binary tree is $(n+1)$

- For any non-empty binary tree with n_0 leaf nodes and n_2 nodes of degree 2, $n_0 = n_2 + 1$.

- The number of edges or branches in a binary tree of n nodes is $n-1$.

Next traversal

pre-order

root → Left → right

in-order

left → root → right

Post-order traversal

left → right → root

Pre-order:

(Root - Left - Right) ← for given Subtree

```
void preorder (struct node * root) {
    If (root != NULL) {
      printf (root → data);
      preorder (root → left);
      preorder (root → right);
    }
```

Post-order without Stack (Left - Right - Root)

∴ Using Stack

1. If T = NULL then
   Write ('Empty tree')
   return
   else P ← T
   Top ← 0

2. [Traverse in Post order]
   Repeat step 5 while true

3. [Descend Left]

   Repeat while P ≠ NULL
   Call PUSH (S, TOP, P)

P ← LPTR(P)

4. [Process a node whose left and right sub-tree have been traversed]

Repeat while S[TOP] < 0

P ← POP(S, TOP)

write(DATA(P))

If TOP = 0 (Have all nodes been processed?) then Return

5. [Branch right and then mark node from which we branched]

P ← RPTR(S[TOP])

S[TOP] ← S[TOP]

Youtube Easy Algorithm

1. Push root to 1st STACK
2. Loop while S1 is not empty
   i) POP top node of S1 & push to S2
   ii) Push (node → left, node → right) to S1
   if (node → left != NULL)

3. Print content of S_2 (post-order).

In-Order

(Left - root - right)

Basic Algorithm

1. Create an empty stack S
2. Initialize current node as root
3. Push the current node to S and set
   Current = Current → left
   until Current is NULL.

4. If Current is NULL and stack is not empty then
   a. POP the top item from Stack
   b. Print the popped item
      set Current = Current → right
   c. Go to Step 3

5. If Current is NULL and stack is empty then we are done.

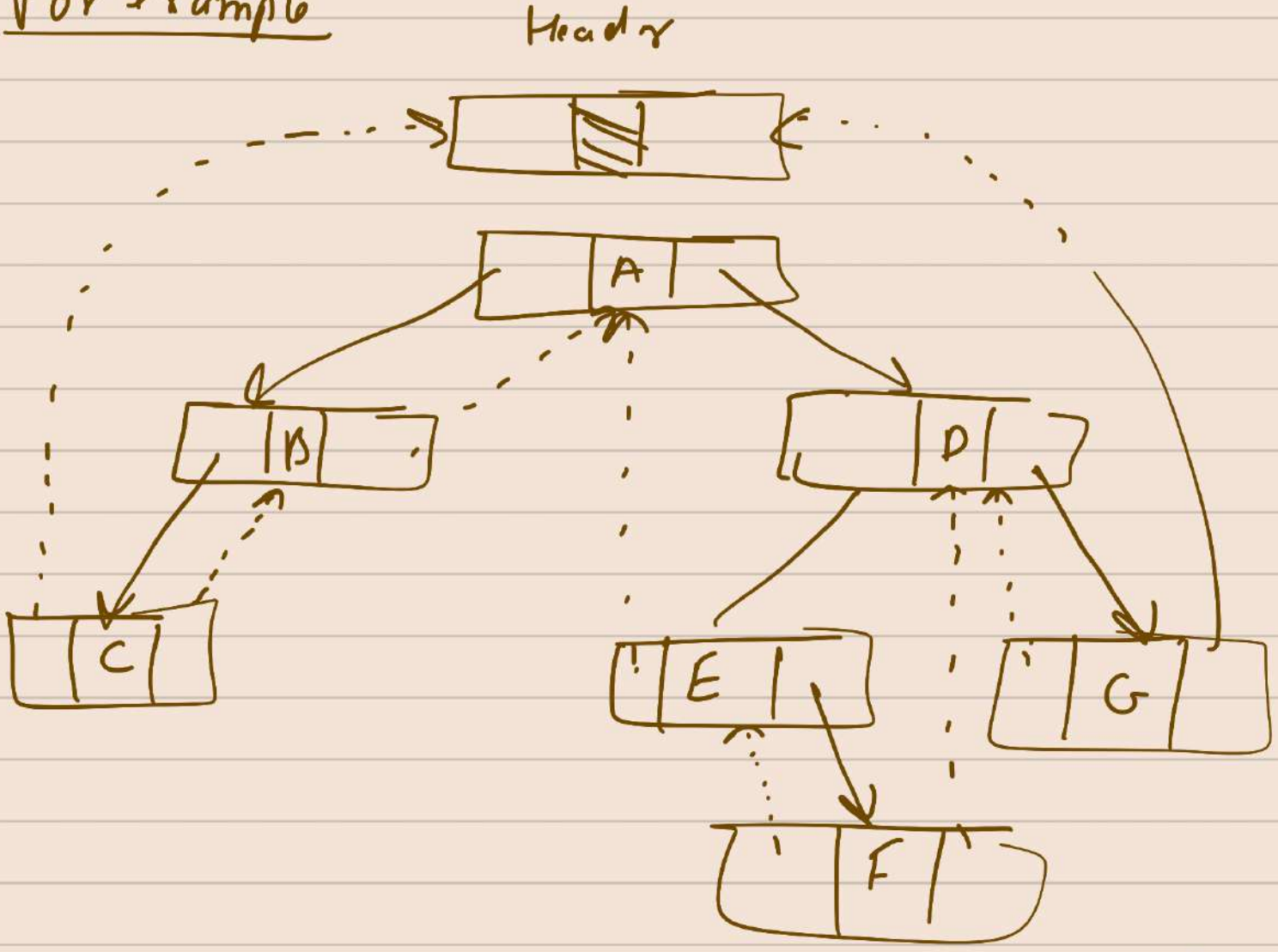
Converse Sub tree

1. RST
2. Root
3. LST

Threaded Storage Representation for Binary tree

* If tree is given
  a. take Inorder of tree
  b. Connect NULL pointer of Nodes to it left in inorder and right to its right Inorder

For Example



Inorder H C B A E F D G H

(- - - ->) ← This lines are thread

Binary Search tree

n = nodes

Max height BST = n - 1

Min height BST = $\log_2 n$

* Also known as Ordered Binary tree
* No more than two child to each node
* Each child must leaf node or not node of another binary Search tree.

* The left sub-tree contains only nodes with keys less than the parent node.

* The right sub-tree contains only nodes with key greater than the parent node.

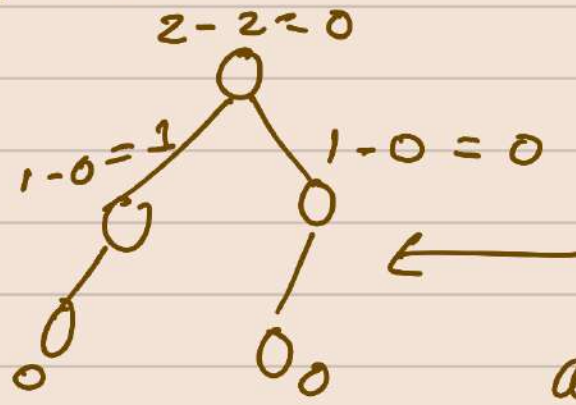
AVL tree

no. of diff. Binary tree can be formed $= \left({}^{2n}C_n \over n+1 \right)$

balance factor = height of left Subtree - height of right subtree.

$bf = hl - hr = \{-1, 0, 1\}$

$|bf| = |hl - hr| \leq 1$



← This is balanced tree because all nodes are in

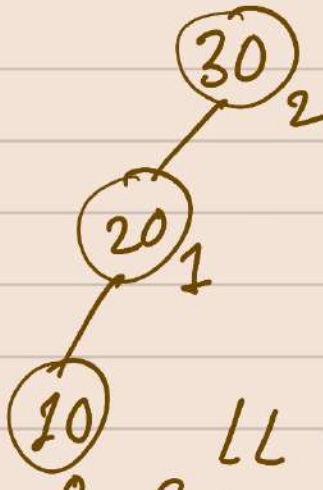
range of {0, 1, -1}

Rotation in AVL tree

Initially

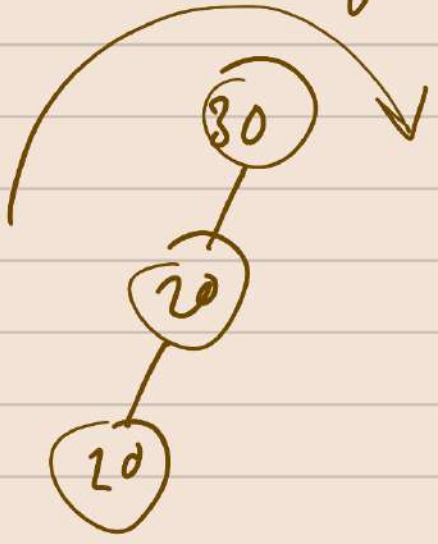
← This balanced

Lets perform insertion

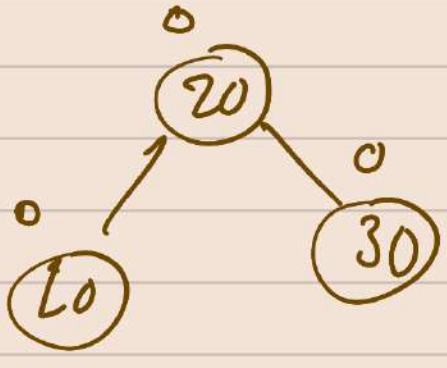


← Now this not balanced tree

LL - imbalance
(left of left imbalance)



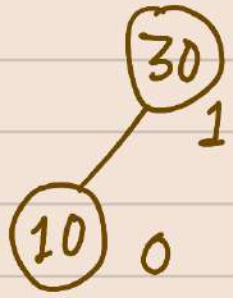
After rotation



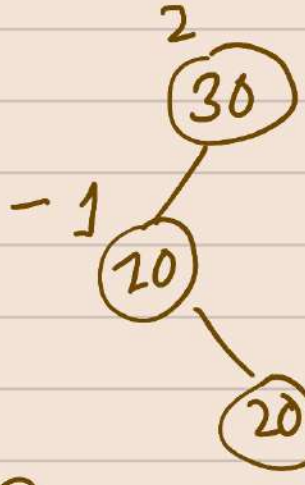
LL-Rotation

#2

Initially



Insert 20

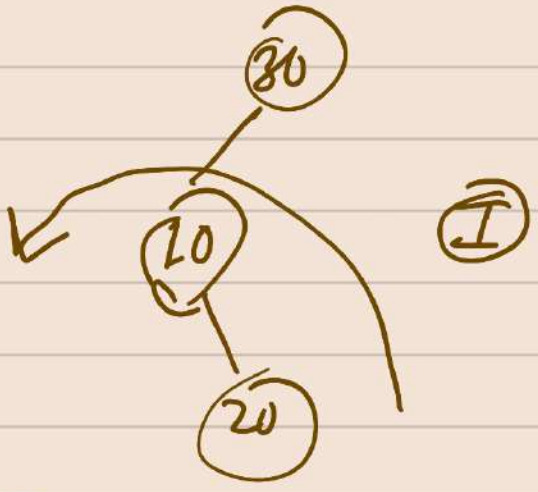


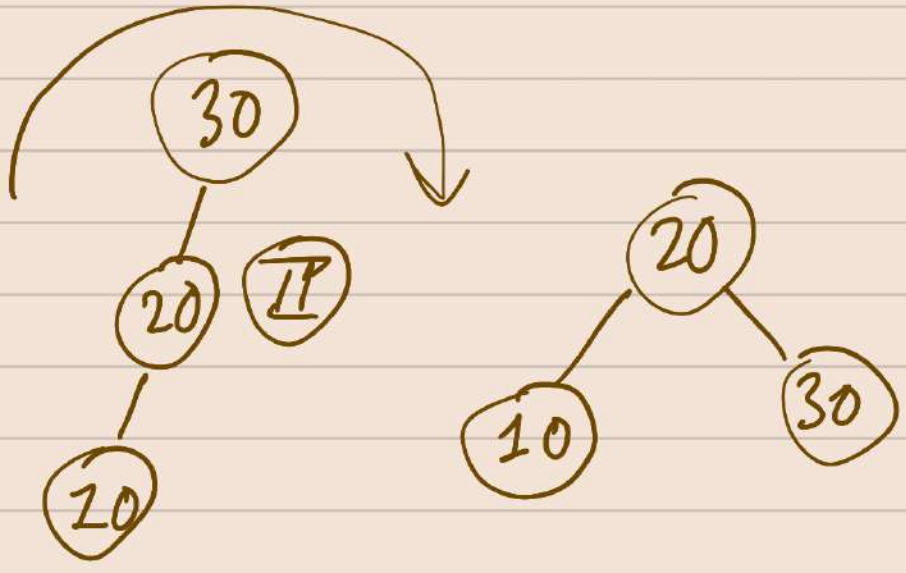
L-R imbalance

(two step rotation)

(Double rotation)

Performing Rotation





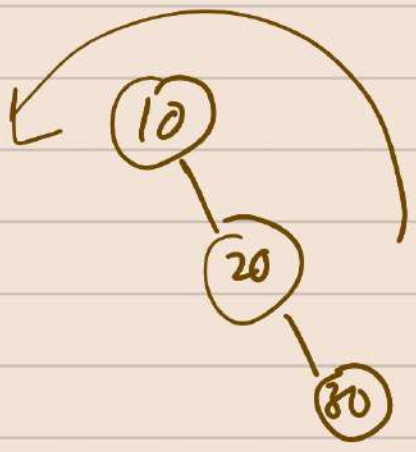
LR Rotation

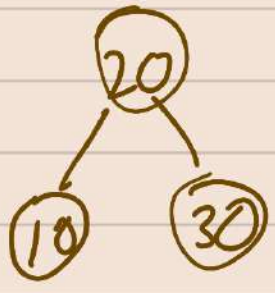
#3

Initially

Insert 30

RR Imbalance

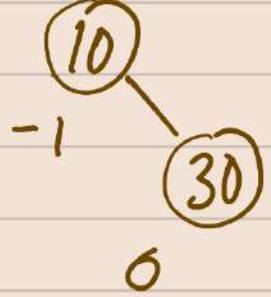




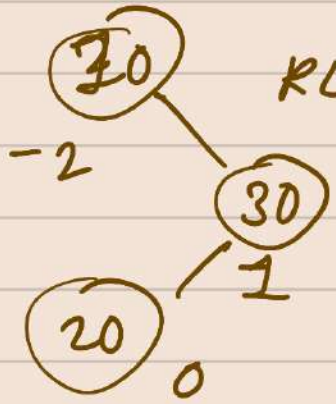
← RR Rotation

#4

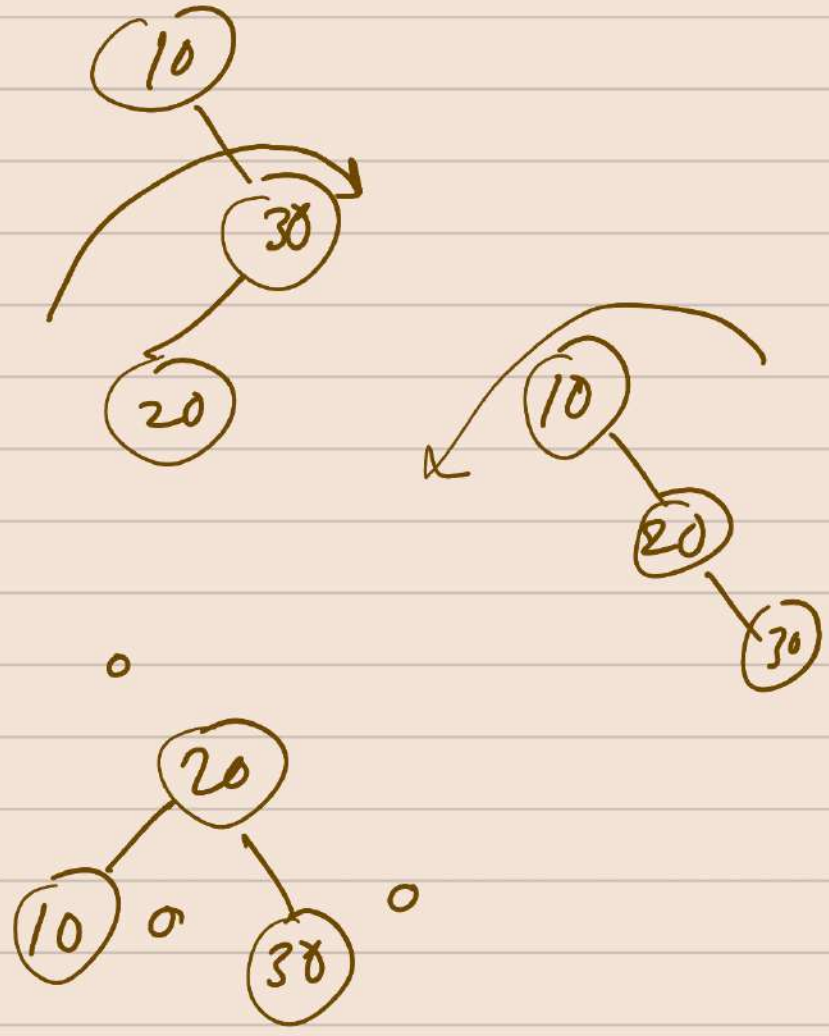
Initially



Insert 20

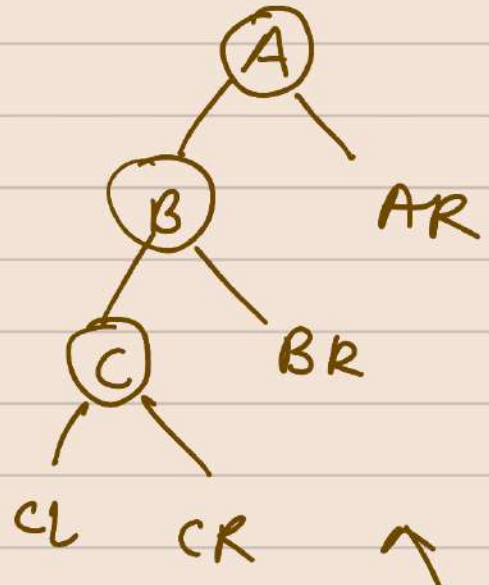


RL-Imbala

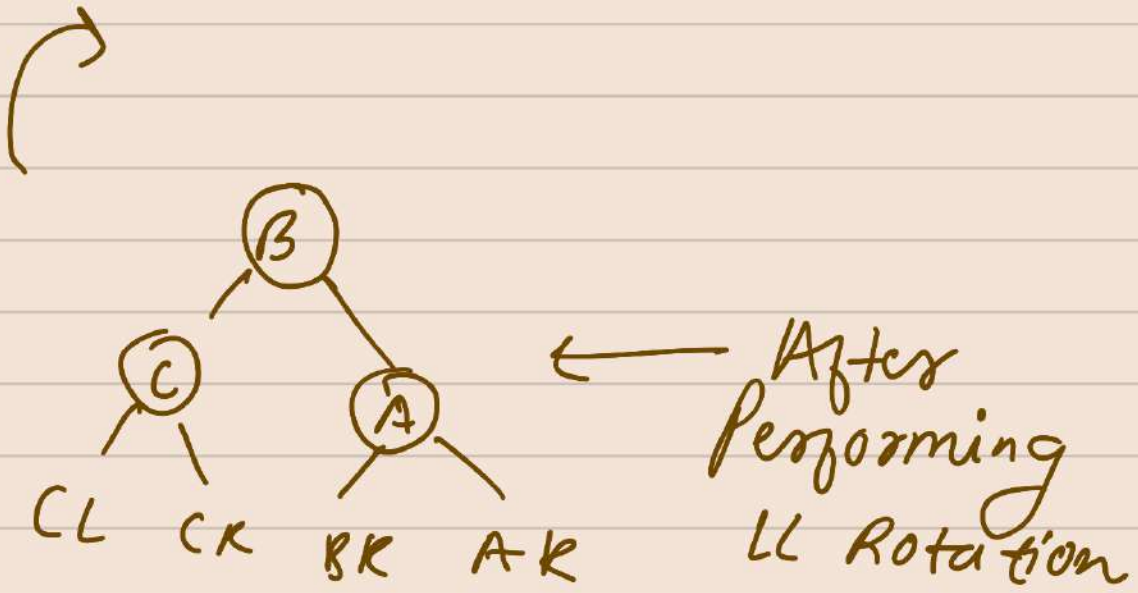


RL rotation

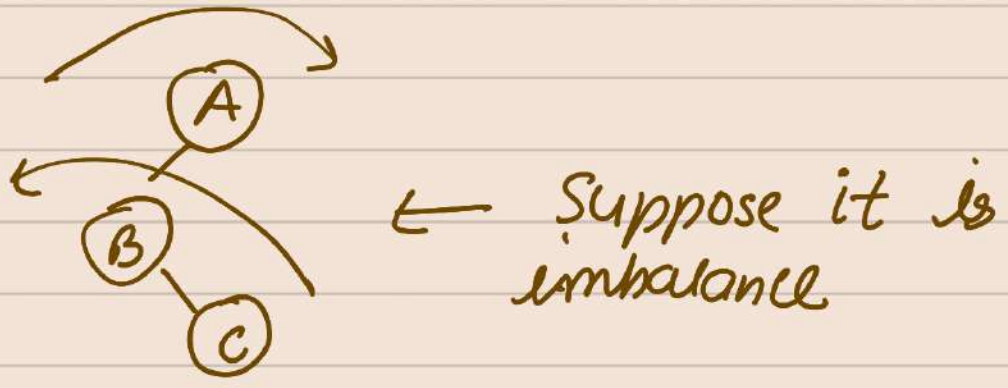
How to Create AVL tree?

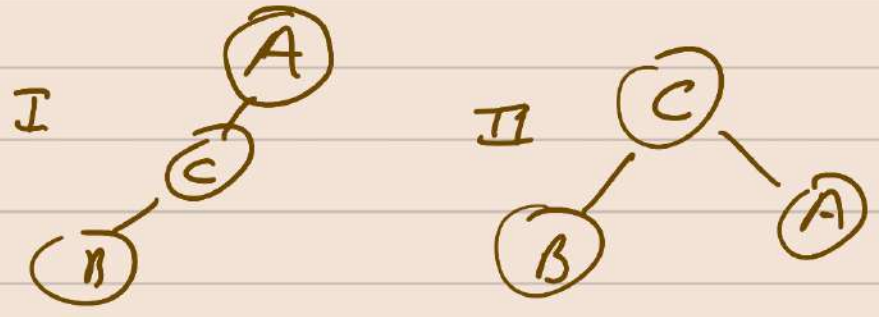


performing LL Rotation
of Imbalance
(Let assume it is Imbal..)

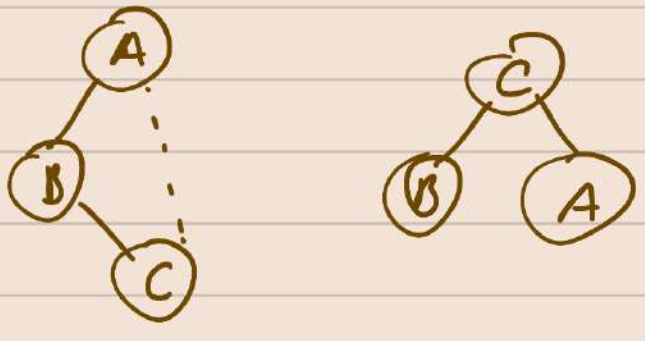


L-R Rotation



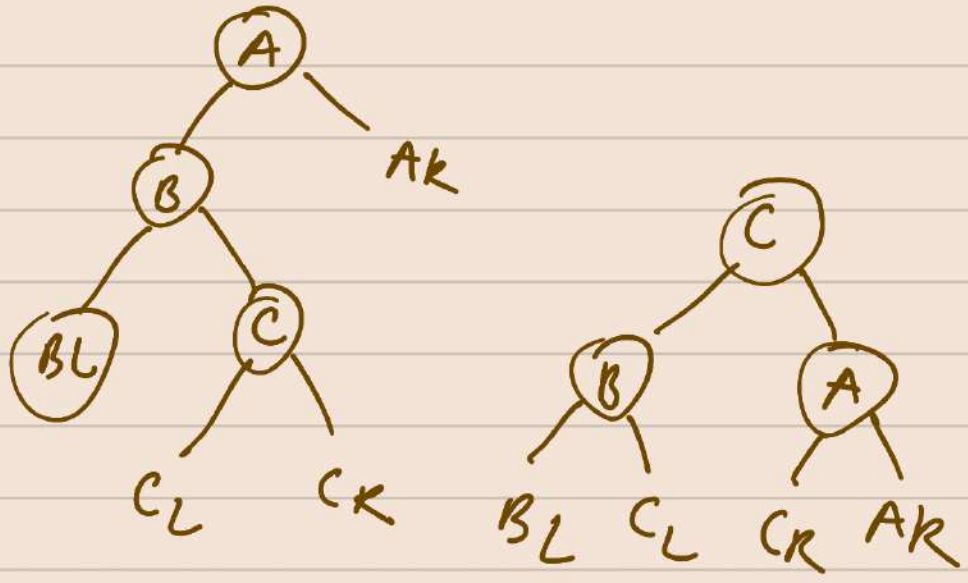


We can show Single Rotatn also



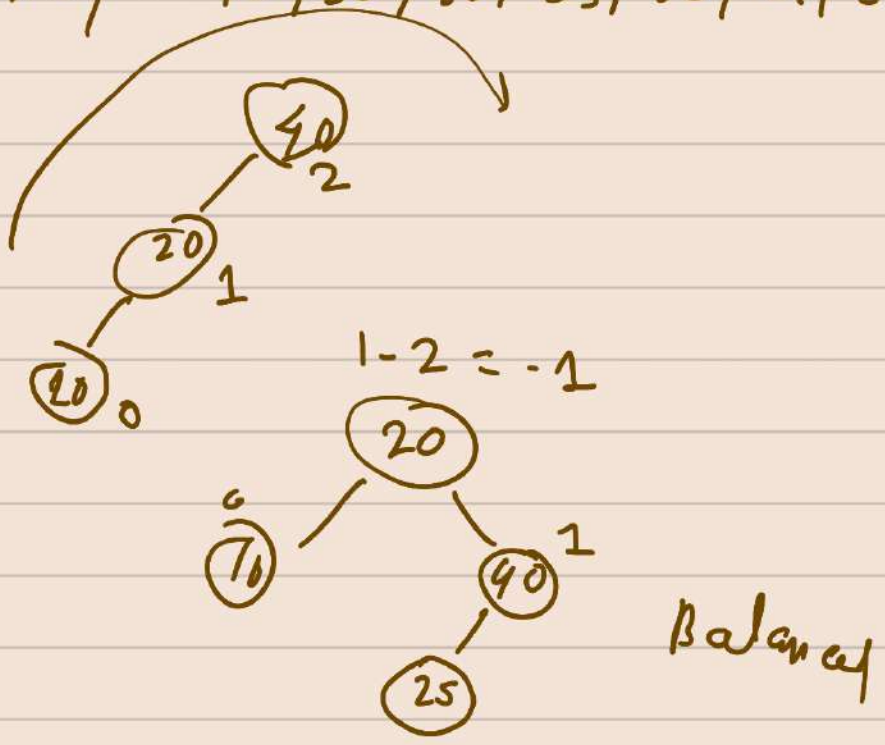
For Biggest tree

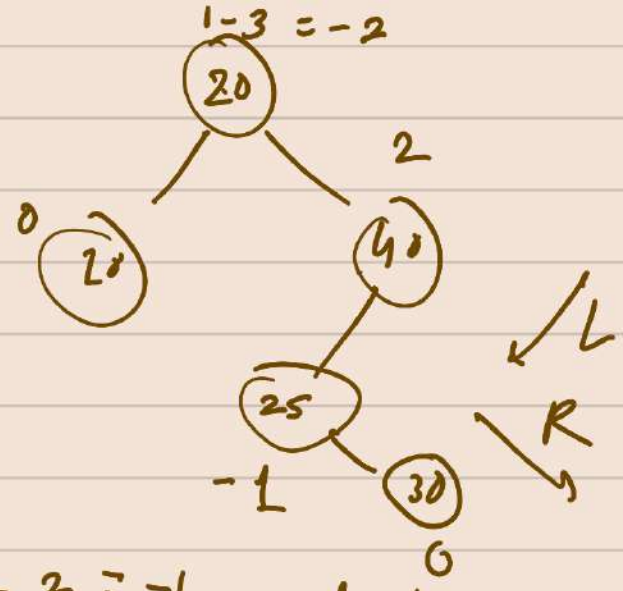
L R Rot.



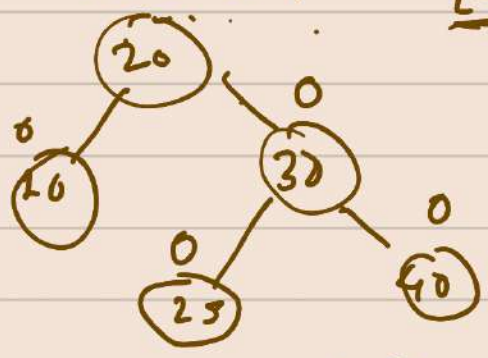
AVL tree Creation

key : 40, 20, 10, 25, 30, 27, 50

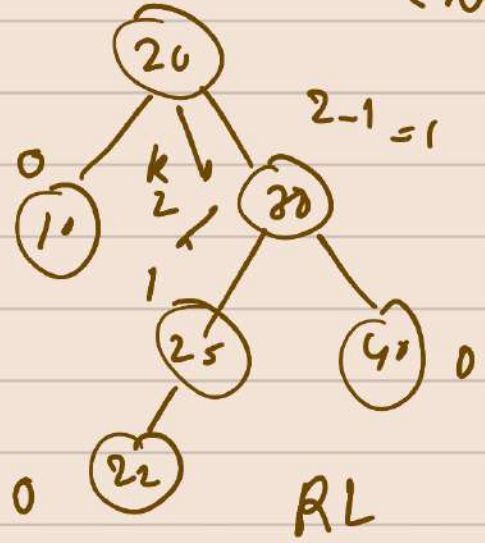




1 - 2 = -1 L - L now

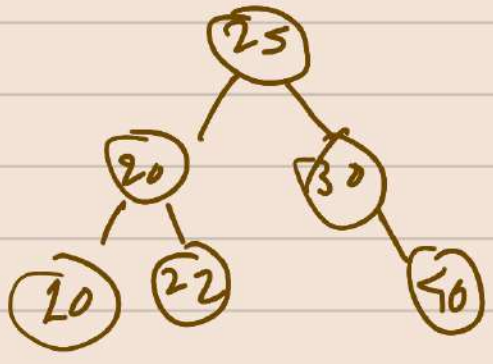


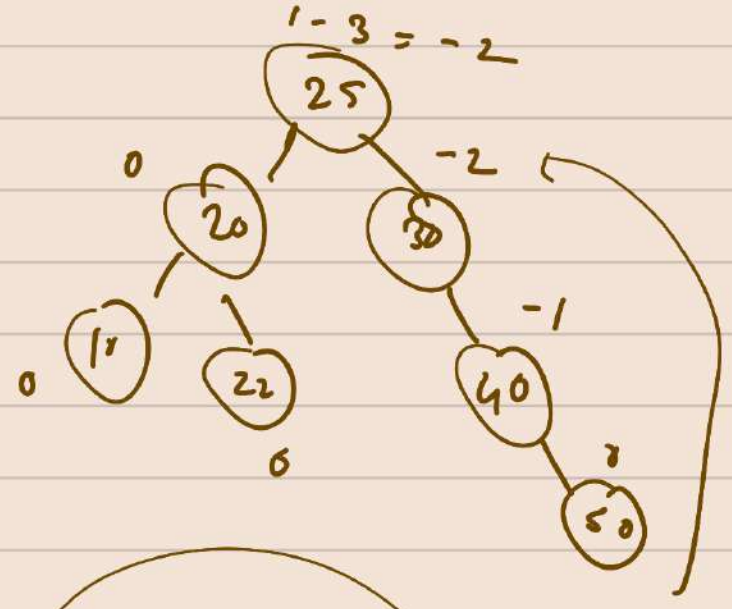
1-3 = -2 (we consider longest distance)

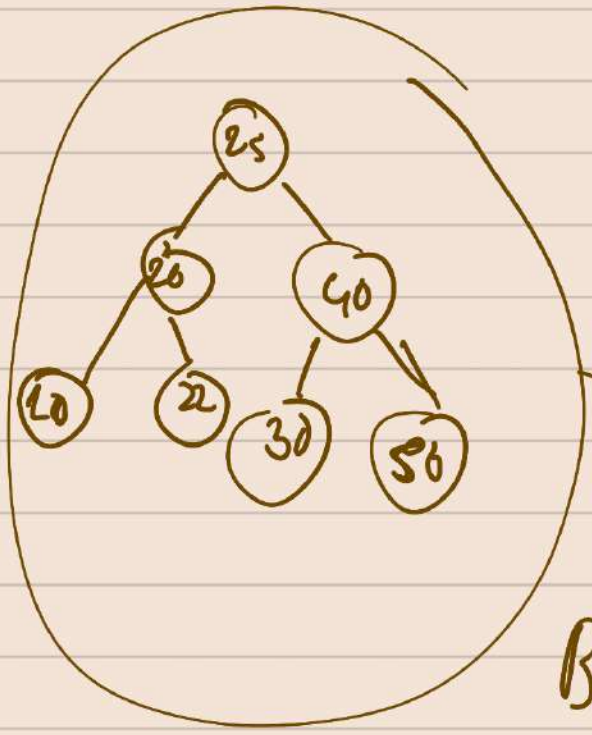


RL

Perform RL

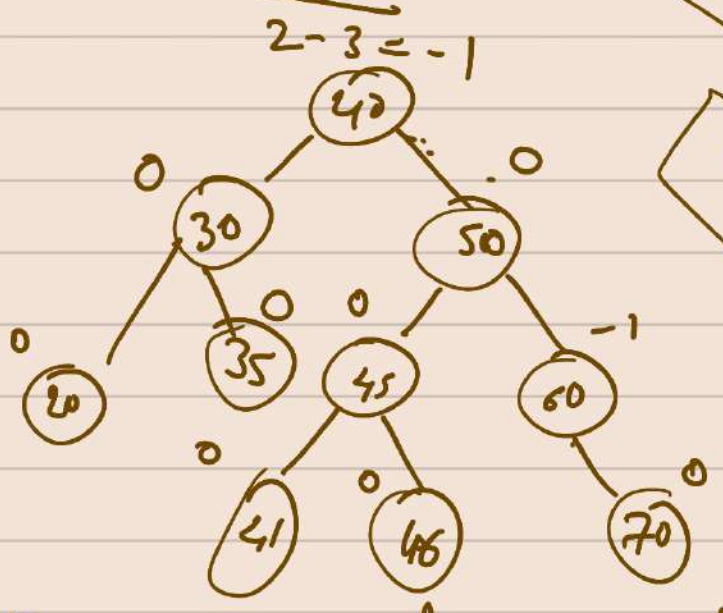






Complete Balanced tree.

AVL tree



O(log n)

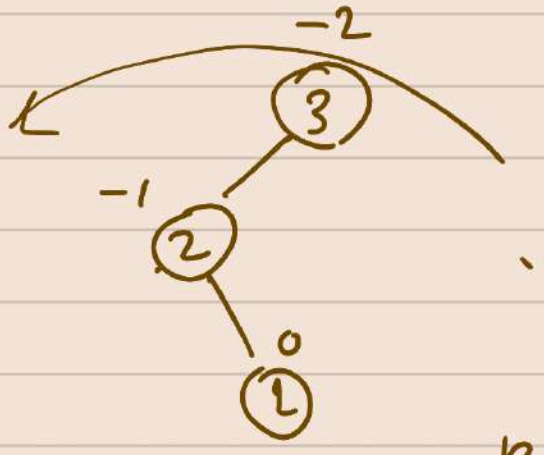
1.44 log n

AVL tree alway maintain log n height

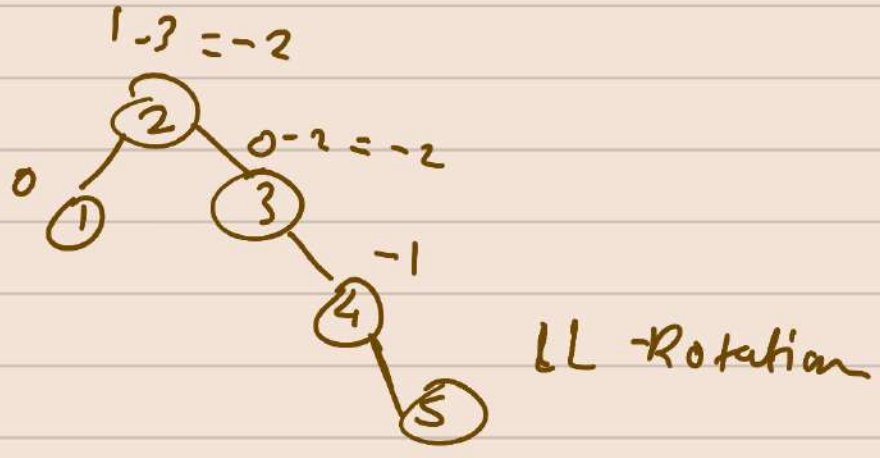
Red Black tree

Example no AVL tree (BOJ)

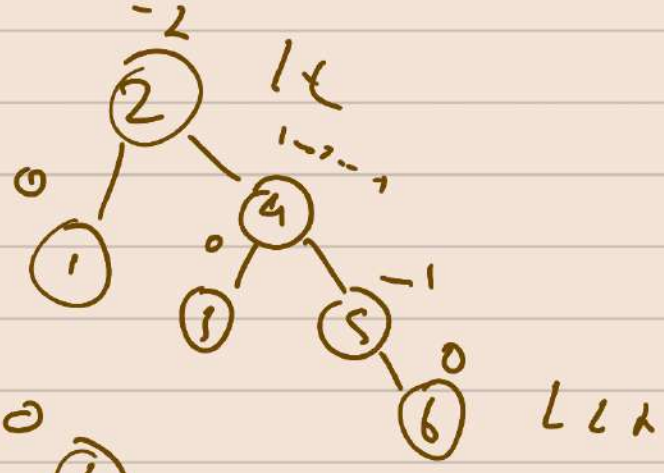
3, 2, 1, 4, 5, 6



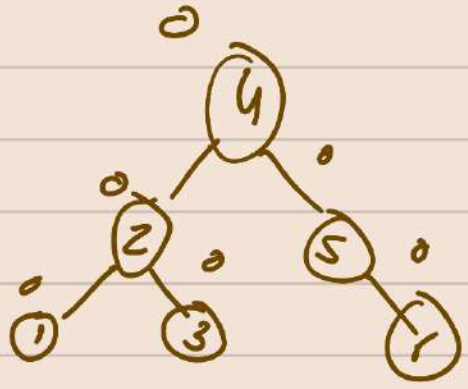
RL



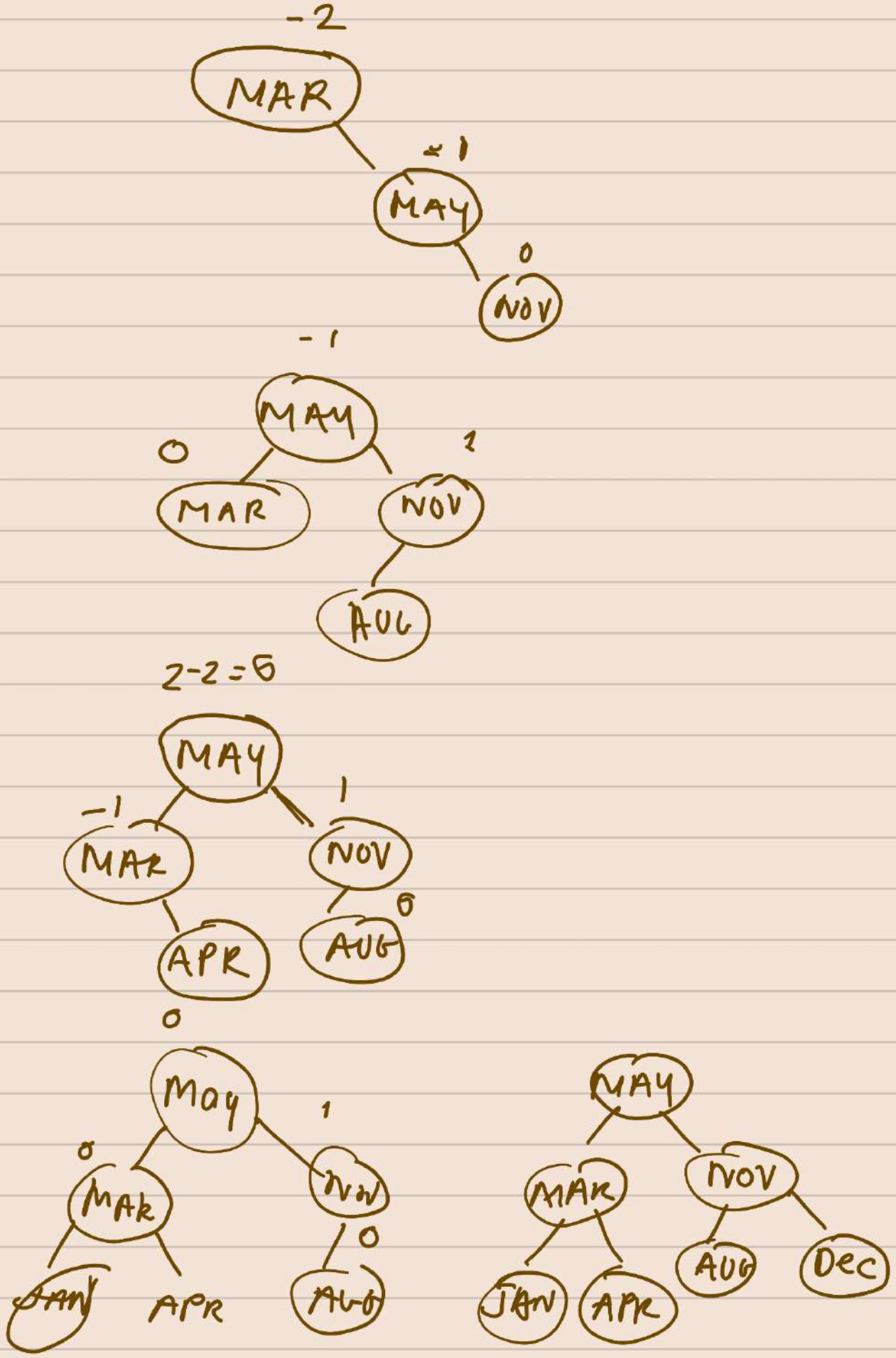
LL Rotation

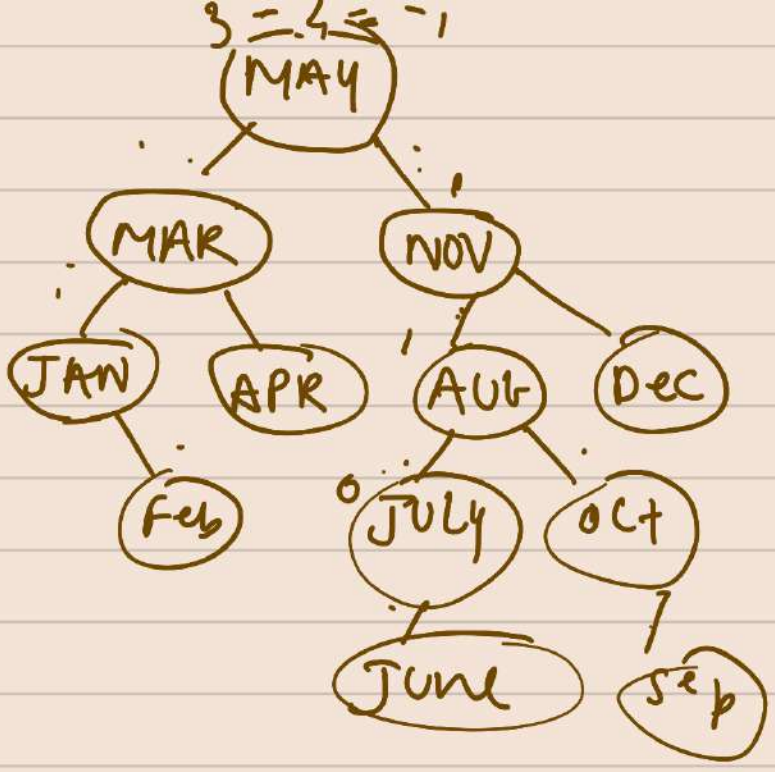


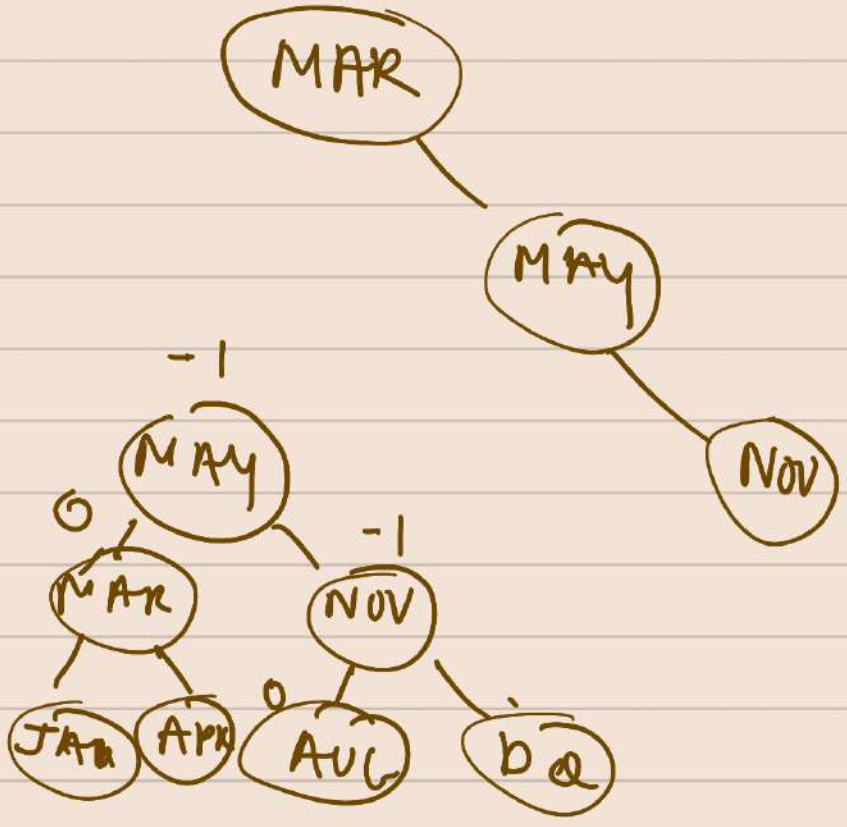
LLR



MAR, MAY, NOV, AUG, APR, JAN, DEC, JULY, FEB, JUNE, OCT, SEP.




MAY
MAR
NOV
JAN
APR
AUG
Dec
Feb
JULY
oct
June
Sep


MAR
MAY
NOV
MAY
MAR
NOV
AUG

CHAPTER 5 Graphs

* Graph is a collection of nodes called as Vertices
* lines connecting two Vertices are called as edges
* Graphs

Directed undirected

* Directed Graphs are the graph which each line as a direction.
* Undirected Graphs are the Graphs in which each lines donot have direction
* Two Vertices are said to be adjacent if an edge directly connects them.

* Paths are Sequence of Vertices
* A cycle is a path of atleast 3 Vertices. Each adj to next
* A cycle is a path of a least three Vertices that Starts ends with the same vertex.
* Loop 
* The degree of vertices is the number of arc or edges leaving the nodes.
  The indegree of Vertices is the number of arc and edges entering the nodes.

* Six operations are done in Graphs
  1) Add a vertex 2) Delete vertex 3) add an edge 4) Delete an edge 5) Find a node 6) traverse the graph

There are two Standart Graph traversal : Depth First and breath First:

★ In depth first traversal, all of node's descendents are processed before moving to an adjacent node.
★ In breath first traversal all adjacent nods are traversal before procesing towords descendents of vertex

Storing of Graphs

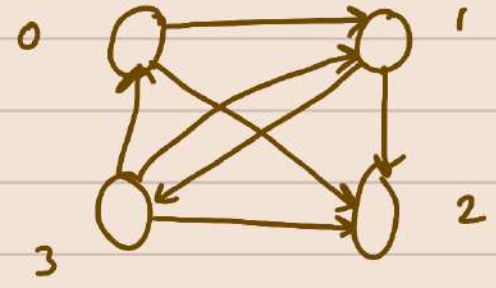
★ Adjancency matrix method
★ Adjancency list method.

Adj- Matrix Method

★ In the adjancency method, we use a vector to store the vertices and matrix to store the edges.
★ In the adjacency list method we use linked a linked list to store the edges.

- A network is a graph whose lines are weighted
- A Spanning tree is a tree that Contains all of the vertices in the graphs.
- A minimum Spanning tree is a spanning tree in which the total weight of the edges is the minimum.

Graph Representation



Sets Representation

Adjacency Sets

Vertex	Sets
0	{1, 2}
1	{2, 3}
2	ϕ

3 {0, 1, 2}

Adjancency table Matrix

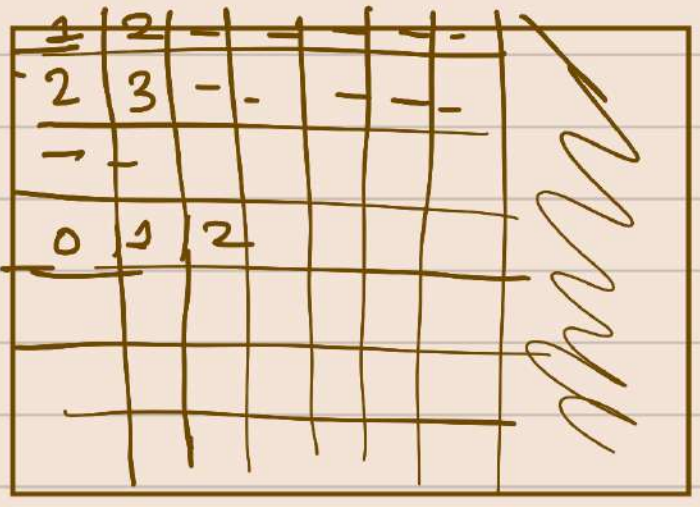
	0	1	2	3
0	F	T	T	F
1	F	F	T	T
2	F	F	F	F
3	T	T	T	F

3// Contiguous List

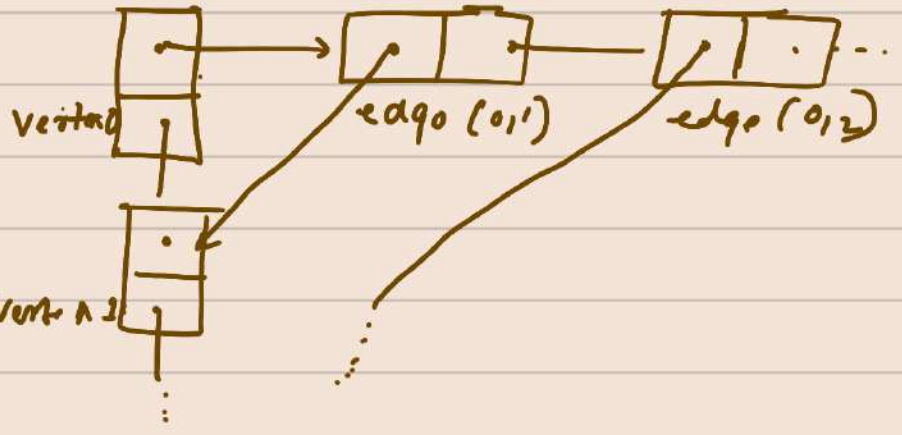
count = 4

Valance	Vertex
2	0
2	1
0	2
3	3
	4
	5
	6

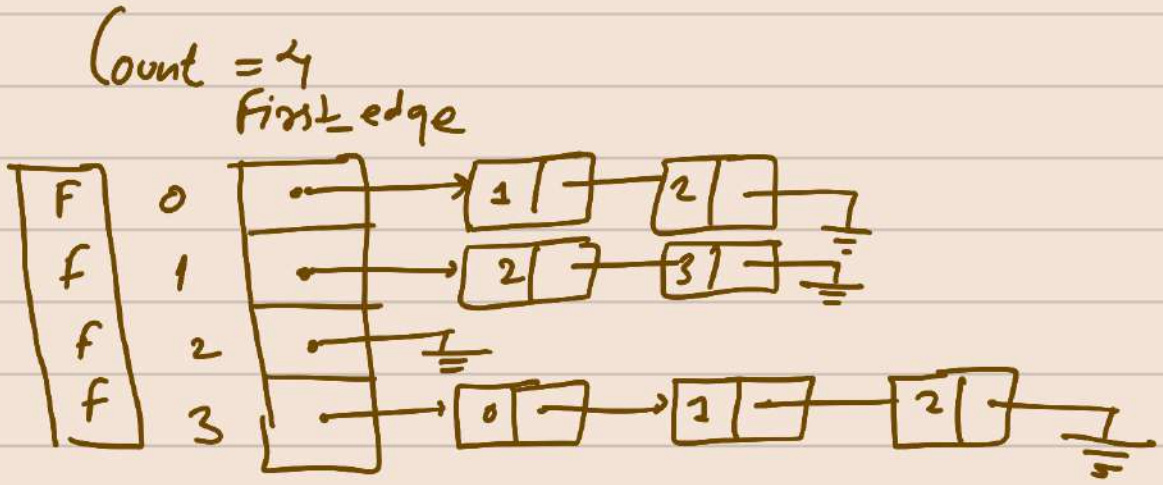
Adjacency list



Linked List (Is length) only for 2



05) Mixed Representation (Hybrid)



Graph traversal Algorithm

DFS Algorithm

Given an undirected graphs $G=(V,E)$ with n vertices and an array visited[] initially set to zero, this algorithm visits all vertices reachable from v. G and Visited[] are global

```
{
    Visited[v] = 1;
    for Each Vertex w adjacent frm v do
    {
        if (visited[w] = 0)
            then DFS(w);
    }
}
```

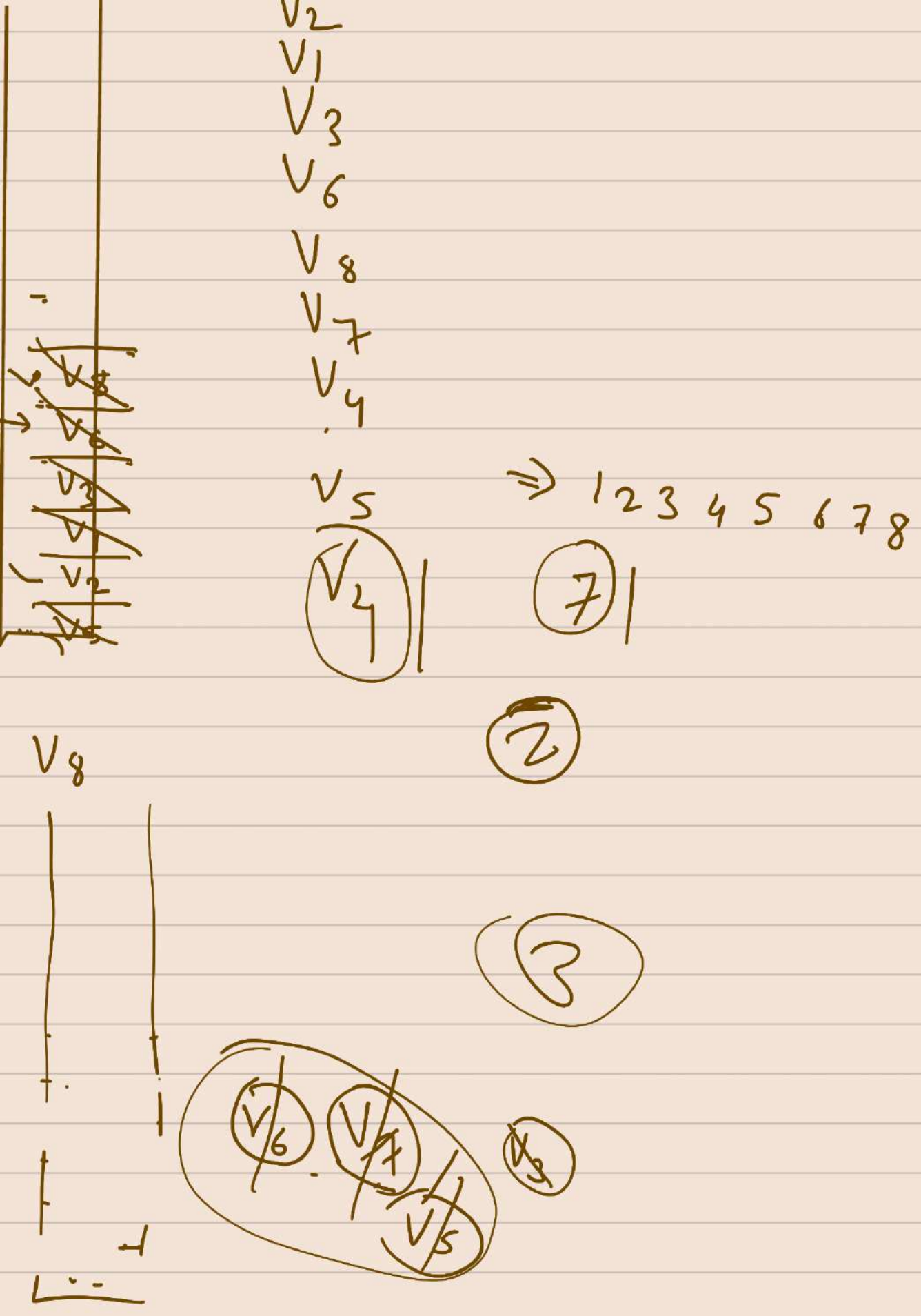
BFS algorithm

Label Vertex V as reached;
initialize Q to be a queue with only V in it;
while (Q is not empty) {

delete a Vertex W from the queue;
Let U be a Vertex (if any) adjacent from W;
while (U) {
if (U has not been labeled) {
Add U to queue;
Label U as reached; }
U = next Vertex that is adjacent from W;

Imp

Level-order traversal: Visit every node on a level before going to a lower (this is also called Breath-first traversal)


V_2
V_1
V_3
V_6
V_8
V_7
V_4
V_5
$\Rightarrow$ 1 2 3 4 5 6 7 8
V_4
7
2
V_8
3
V_6
V_7
V_5
V_1 V_2 V_3 V_4 V_5 V_6 V_7 V_8

Hashing #6 (Efficiency of Sequential list is O(n))

- In a hashing Search the key, through an algorithm transformation, determines the location of the data. It is a key-to-address transformation.

Hashing functions

* There are Several hashing Functions:

Direct, Subtraction, Modulo division, digit Extraction mid Square, Folding, Rotation and Psedorandom Generation.

1) **Direct Hashing:**

address are direct keys without algorithm manipulation.

2) **Subtraction hashing:** The key is transformed to an address by Subtraction a fixed number from it.

3) **modulo-division hashing**

the key is decided by the list size, recommended to be prime number and the remainder plus 1 is used as the address.

4) Digit-Extraction hashing: Selected digits are extracted from key and used as an address.

5) Mid Square hashing: The key is squared and the address is selected from the middle of the result

6) Fold shift hashing: The key is divided into parts whose size match the size of the required address. Then the parts are added to obtain the address.

7) Fold boundary hashing: The key is divided into parts whose size matches the size of the required address. Then the left and right parts are reversed and added to the middle part to obtain the address.

8) Rotation Hashing: The Right most digit of the key is rotated to the left to determine an address. However, this method is usually used in combination with other method.

9) Pseudorandom Generation Method: The key is used as the seed to generate a pseudorandom number. The result is then scaled to obtain the address.

Collision Resolution Techniques

- Collision occurs when a new keys is hashed to an address that is already occupied.
- Two general methods are used to solve Collision Open addressing and linkedlists (chaining).
- Open addressing method can be subdivided into linear probe, quadratic probe, pseudorandom and rehashing.
- In the linear probe method when the collision occurs, the new data will be stored in the next available address.

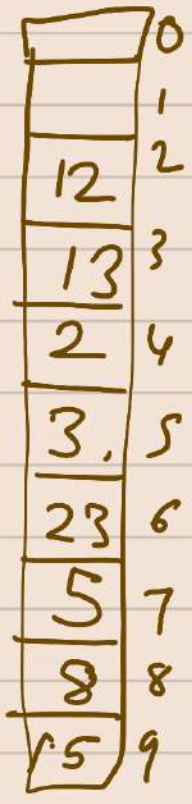
Separate chaining method: A Separate linked list is maintained for each set of colliding Records.

Examples

⇒224562 total size = 19

(1) Module Division

224562 % 19 =



$h(k) = k \bmod 10$

$12 \bmod 11$

87	0
11	1
4	2
36	3
92	4
71	5
13	6
14	7
	8
	9
43	10