6/DEC/2022

STACK

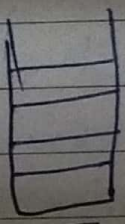
LIFO

Top (open end).
Bottom (closed).

Insertion : Push (element) → on Top.
Deletion : Pop( ) → from Top.

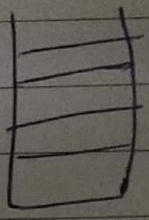
PUSH :

```
Top = -1
Push (A)
if (Top != size-1)
    Stack [0] = A
    Top++
else
    stack is full / stackoverflow.
```



Top = -1

Top == size-1
overflow



Top = size

Top == 0
overflow

PEEK → it prints the top element in stack..

Eg: Backtracking ⇒ Puzzles, games, undo

8/12/22.

Linked List / Stack:

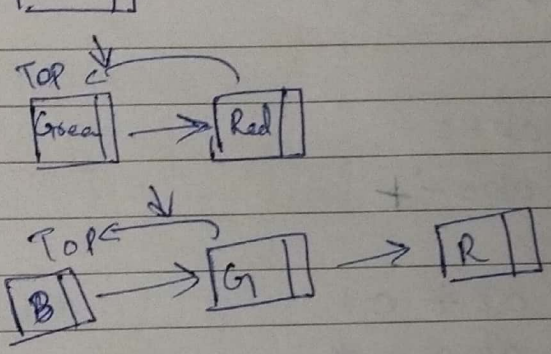
→ Top == null {Empty stack}
- Insertion (this condition makes normal insertion)
- Deletion (Stack underflow).

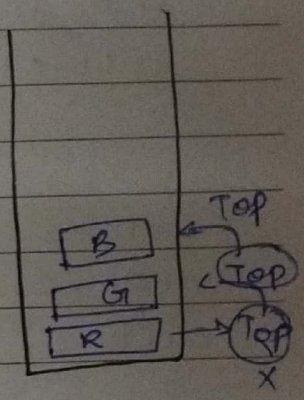
→ Top^x → count == n.
- insertion → (Stack overflow).

Top == null

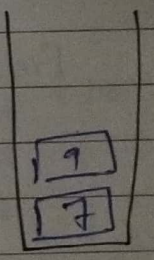
Insertion at start

Push()





x = 7
y = 9
push(x)
push(y)
x = pop() => x = 9
y = pop() => y = 7
print (x, y)
9, 7



Finals

r = 64 al ax eax
e = 32 ah bx rax
x = 16 bl 64
y = 8 bh
h

✓

20/12/22

Infix (Human)
In-between operands
operand op operand
a op b
requires precedence
a + b
(high level)
a+b+c
(a+b)+c
a+(b+c)

Postfix (Computer)
after operands
operand operand op.
a b op
doesn't
a b +

Prefix
before operands
op operand operand
op a b.
doesn't
+ ab.
(low level)

(a+b)+c → ab+c+
a+(b+c) → +a+bc

a+b-c
(a+b)-c → ab+c-
a+(b-c) → abc-+

a*b+c → ab*c+
a*(b+c) → abc+*

1. () Precedence
2. ^
3. * /
4. + -

& Association
L → R.

a^b^c (R→L)
a^(b^c)

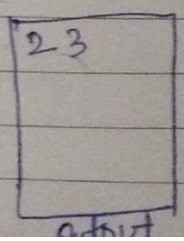
① 2*3+4/5
23*45/+
→ 2345/*+

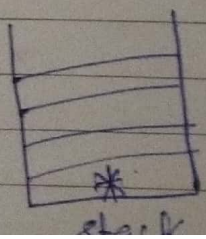
23*
45/
01+

23^

✓✓ 23*45/+ (Postfix)

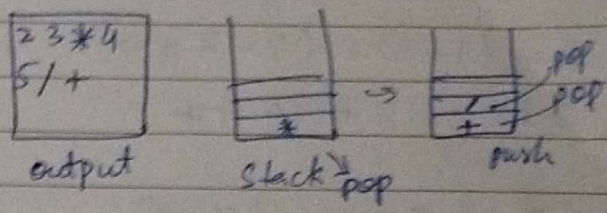
① variable
② operator ← stack



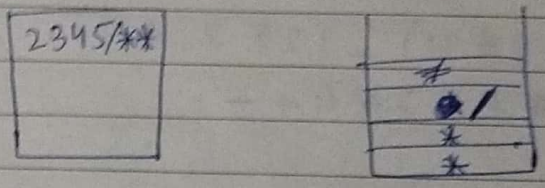


① 2*3+4/5 -> 23*45/+ ✓

① variable
② operator

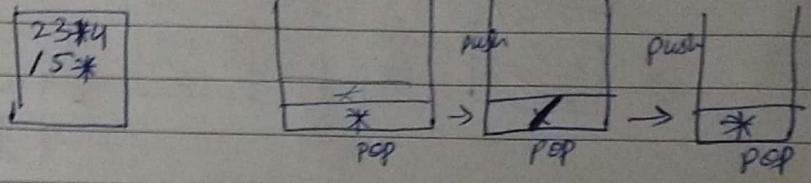


② 2*3*4/5 ⇒ 2345/**



23*4*5/

2*3/4*5



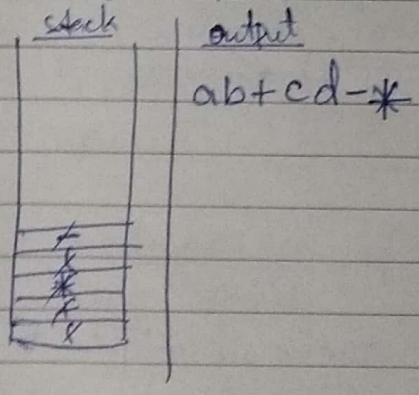
23*4/5*

3) 2+3*4/5

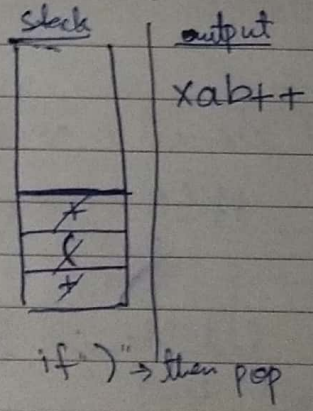
234*5/+

22/12/22.

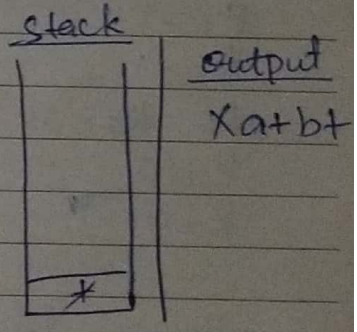
(a+b)*(c-d)



X+(a+b)



X+at b.



① (a+b)+(c+d)
ab+cd++

② ((a+b)*c)+d *
ab+c*d+*

③ {a*(b+c)/d*(4-z)} <
abc+*d/4z-*

29/12/22.

Expression Evaluation:

1. $5 + (1-3) + (2+7*9)$. $= 68$

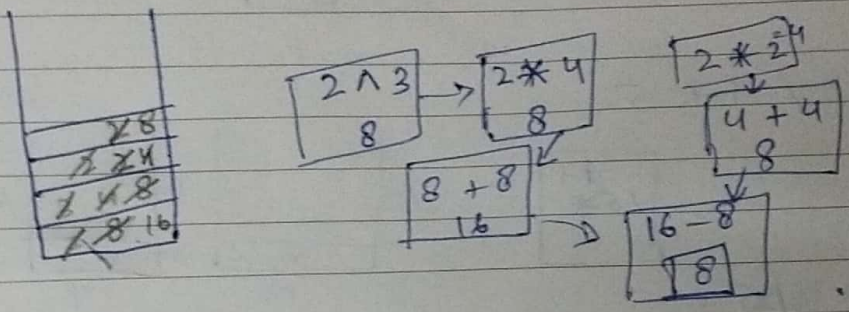
   5 1 3 - + 2 7 9 * + +

2. $(5*3)*2 + (4-3)$. $= 31$

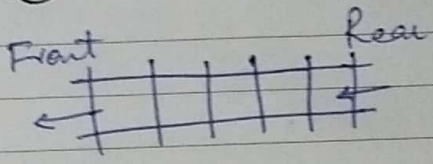
   5 3 * 2 * 4 3 - +

3. $(2\wedge 3)+(2*4)-(2*2+4)$. $= 8$

   2 3 ^ 2 4 * + 2 2 * 4 + -



QUEUE: FIFO



Delete → Dequeue( ) → Front / Head

Insert → Equeue(x). → Rear / Tail.

Peek (.).

(normal, circular, priority) queues.

when queue is empty.

F = -1

R = -1

when there is one element in queue.

F = 0

R = 0.

```
if(Q == empty)
    F++;
    R++;
```

when we delete last element from queue, front & last ~~element~~ will reset

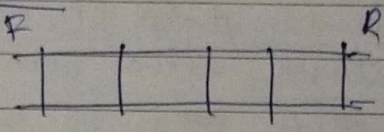
F = -1

R = -1

3/1/23

Types of Queue:
① Normal
② Circular
③ Priority.

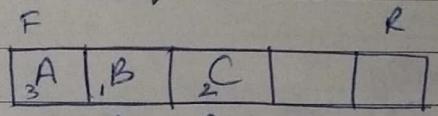
Priority Queue:



A B C D
3 1 2 5

- if element on front has lower priority then the element to be inserted then first dequeue all elements in queue then enqueue the high priority element and enqueue all elements in the same sequence they dequeued - If element on front has higher priority then just enqueue new element from rear.

A => data
3 => priority.



A B C D
3 1 2 5

D_5 A_3 B_1 C_2

DQ, DQ, DQ
(A) (B) (C).

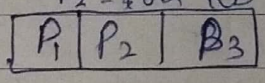
DQ => delete from front. the ENQ.

ENQ(Pointer, Data) => for normal queue.

ENQ(Pointer, Data, Priority => for priority queue.
↓ sorted insetion.

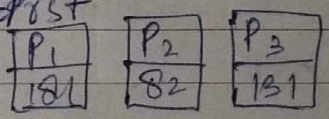
P_1 = 181 KB, P_2 = 182 KB, P_3 = 131 KB.

normal



shortest job ~~first~~ first

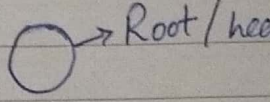
Data priority



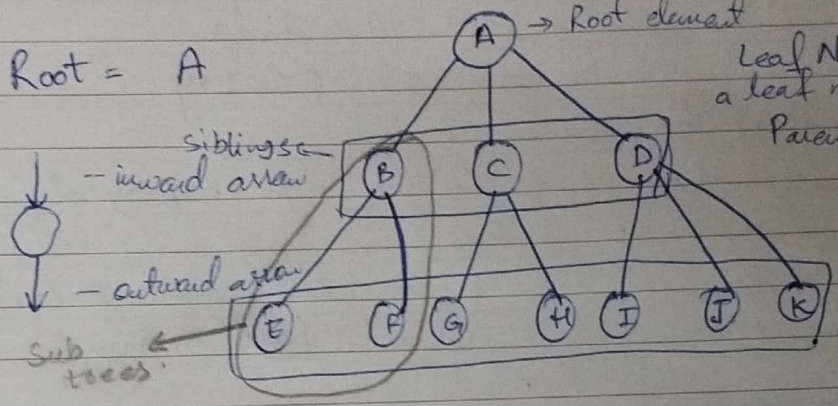
03/01/23

NON-LINEAR DATA STRUCTURES

1) TREE:



- We are mapping non-linear data structure onto linear data structures either array or linked list because we have no other way to save them.



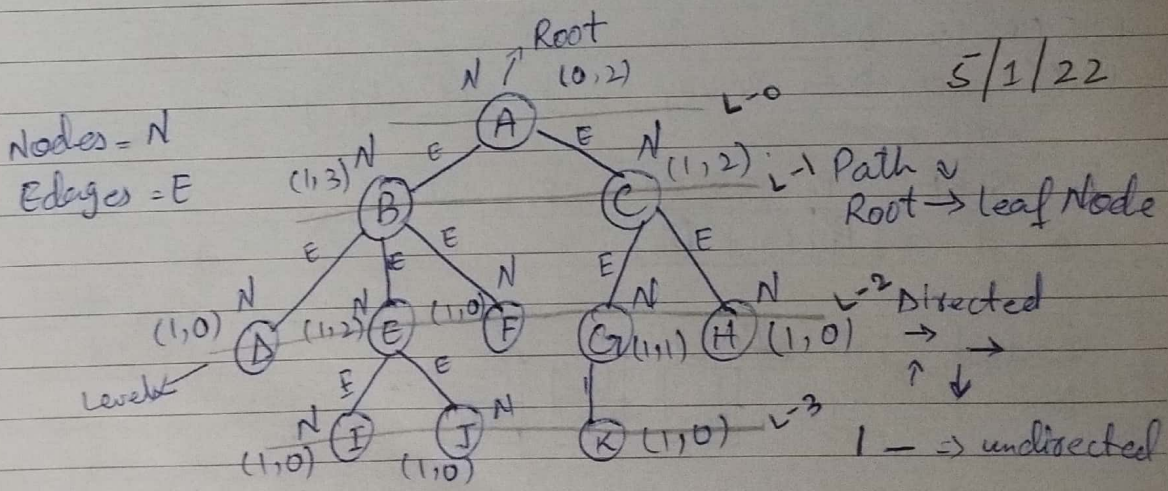
Leaf Node = E, F, G, H, I, J, K
a leaf node can't be child.

Parent node
B → (E, F)
C → (G, H)
D → (I, J, K)
A → (B, C, D)

∵ siblings have same priority.
- parent an child know each others address.

Paths ⇒ A → B → E
A → B → F
A → C → G
A → C → H
A → D → I
A → D → J
A → D → K.

5/1/22



Nodes = N
Edges = E

Path ⇒ Root → Leaf Node

Directed → ↑ ↓

1 — ⇒ undirected

Paths
1. A - B - D
2. A - B - E - I
3. A - B - E - J
4. A - B - F
5. A - C - G - K
6. A - C - H

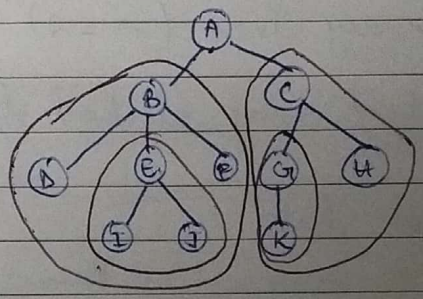
- total paths = total leaf nodes.

! only root node can have incoming 0 or else there is issue in tree

incoming arrows
outgoing arrows
(I, 0)

- Root node will have incoming = 0, otherwise not a tree or tree is not implemented correctly.
- The nodes that have outgoing = 0 are leaf nodes.
- Internal nodes are non-leaf nodes which have ~~both incoming and outgoing~~ - one or more child nodes.
- All nodes in same level are peer
- Levels are counted through edges.
- maximum no. of level is max. no. of edges.
- Siblings have same parents, peers have same levels.

→ Subtrees.



Ek node sy jitni edges niklain woh ek subtree ban jati hai

- ~~Depth~~ Degree of tree : max. no. of nodes

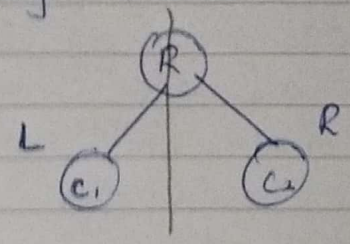
Degree (A) = 2.
Degree (B) = 3
Degree (C) = 2
Degree (D) = 0
Degree (E) = 2

max. outgoing edges is the degree of trees.

- Depth = Level hence can be counted through edges.
  Depth of tree Root → Leaf (max.) {:: max level}.
  Depth of a particular node is its level.
  Depth (C) = 1
  Depth (F) = 2
  Depth (I) = 3
- One node can't have two incoming arrows ✗
- Height of Tree
  → from leaf to Root (count from every leaf to root, the maximum count will be its height).
  → from leaf to node ⇒ H(E) = 1
- n <= 2 (Binary)

• n <= 2 (Binary)
Binary Trees.

: R = Root
C = Child.



10/01/2023.

• General Tree • n-ary tree (n => no of children).
↓
Binary Tree (n = 2)

it can be implemented using linked list & arrays.
→ 1D array
→ Doubly linked list
↓
node {
data.
next / Right child
prev / Left child.
}

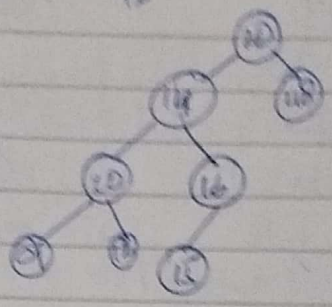
Binary → sorted

Operations
1. → Construction of Tree.
2. → Insert Node (value).
3. → Delete Node (value)
4. → search value
↳ min value ,
↳ max value

LC < R < RC

Construction of Binary Tree.
→ node by node.
→ if root is greater than element (R > E), add element on left.
→ if root is lesser than element (R < E), add element on right.

20, 14, 48, 12, 10, 15, 9

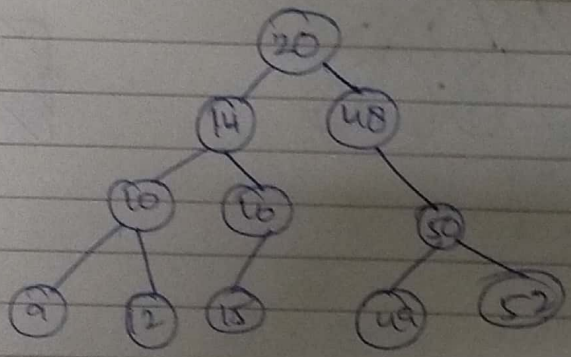


BST
Binary Search Tree.

```
Search() {
  while (current != null) {
    if (current.data == key) {
      return child
    }
    current.
  }
```

① Leaf node delete → Simple deletion
  ① Delete 12 ⇒ 10. Right child = null

Leaf (current.RC == null && current.LC == null)
② Node with one child (connect child to parent's parent)
③ Node with two children



deleting any node.
max value from left sub tree.
or
min value from right sub tree

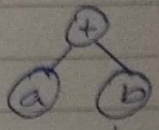
Tree min values are on left most side
Tree max values are on right most side.

17 01 23

Tree Traversal Techniques.

① Infix ② Postfix ③ Prefix.

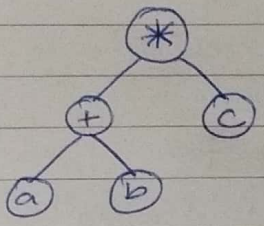
Infix	Postfix	Prefix.
• a+b	• ab+	• +ab
• L Rt R	• L R Rt.	• Rt L R
• In traversal	• Post traversal	• Pre traversal.



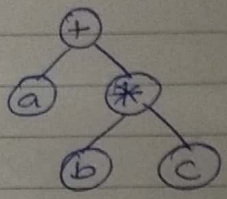
expression tree

a + b

Left Root Right

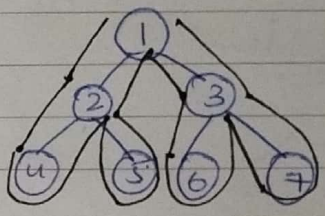


(a+b)*c.

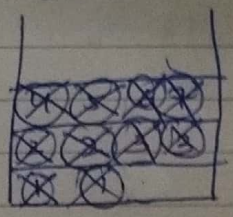


a+(b*c)

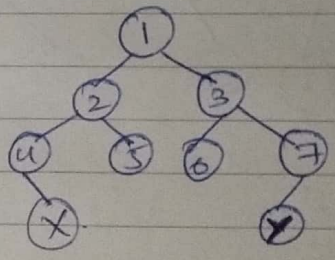
• In expression trees
  └ leaf are operands



stack



Postfix = 4526731.
Prefix = 1245367

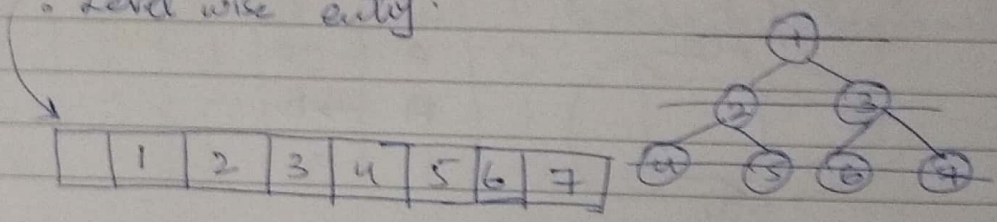


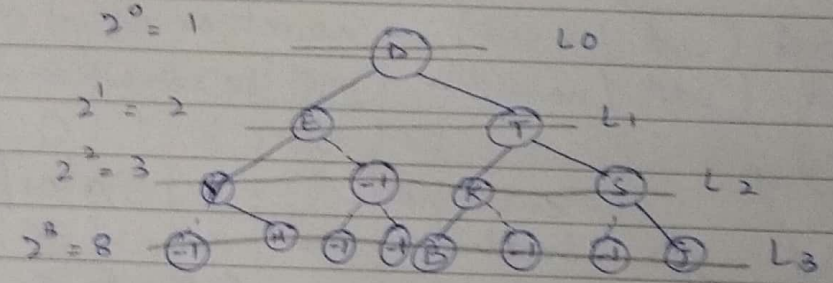
Post fix = X4526Y731
Prefix = 124X5367Y

Array Representation of Binary Tree

Left Child (k) = k * 2
Right Child (k) = k * 2 + 1
Parent (k) = k/2

- a[0] # of elements in tree (Total)
- Level wise entry.





$2^n = m \Rightarrow n$ = level, m = no. of nodes that should be present on that level.

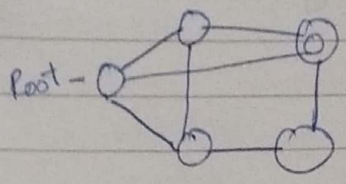
	D	E	T	Y	-1	K	S	-1	H	-1	-1	B	-1	-1	J

skew tree (jhuki hui tree)
└ handled by tree balancing

- Tree is a special form of a graph in which there is only one inward arrow.
  1. Incoming → 1 arrow
  2. No cycle formation (Loop).

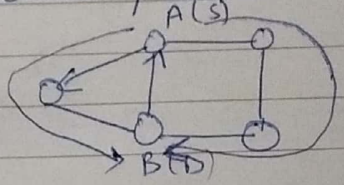
GRAPH:

2D array.



- multiple incoming & outgoing
- no head
- Source, Destination.
- Direction matters

- connections are edges.
- shortest distance / path → cost involved.
- undirected / directed.



src dest
A to B
A → B.

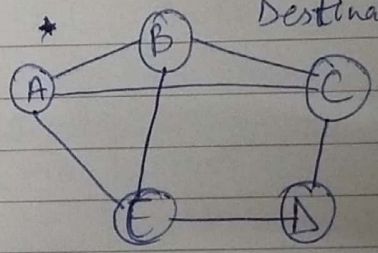
- undirected (cost / no cost)
- directed (cost / no cost).

no cost or equal cost will be considered as 1.

19/01/2023

Source node
Destination node

① Starting point = Root



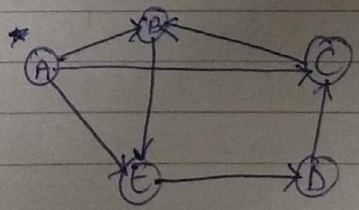
A = Src C = Dest

- A - B - C
- A - C
- A - B - E - D - C
- A - E - B - C
- A - E - D - C

Edge is a connection b/w two nodes/vertex.

Graph = {V, E}
V → vertex, E → Edges.

Directed
Undirected (Bidirectional).



A = Src C = Des

- A → C
- A → B → E → D → C
- A → E → D → C

Mixed Graph => both directed & undirected.

optimal => min, max.

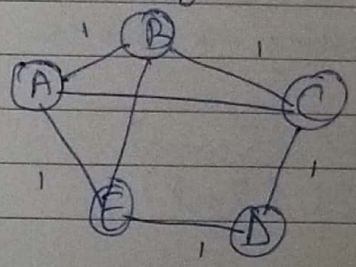
weighted (edges)
unweighted

Shortest path algorithms
↓
kruskul, prims, djextra.

① Adjacency Matrix ② Indices Matrix

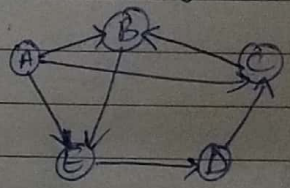
	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	0	1
C	1	1	0	1	0
D	0	0	1	0	1
E	1	1	0	1	0

Bidirectional and unweighted



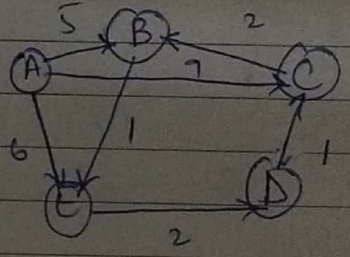
	A	B	C	D	E
A	0	1	1	0	1
B	0	0	0	0	1
C	0	1	0	0	0
D	0	0	1	0	0
E	0	0	0	1	0

Directed and unweighted



★	A	B	C	D	E
A	0	5	9	0	6
B	0	0	0	0	1
C	2	0	0	1	0
D	0	0	1	0	0
E	0	0	0	2	0

Directed & weighted



$9 = A \xrightarrow{9} C$

$9 = A \xrightarrow{5} B \xrightarrow{1} E \xrightarrow{2} D \xrightarrow{1} C$

$XA \xrightarrow{6} E \rightarrow B \rightarrow C$

$9 = A \xrightarrow{6} E \xrightarrow{2} D \xrightarrow{1} C$

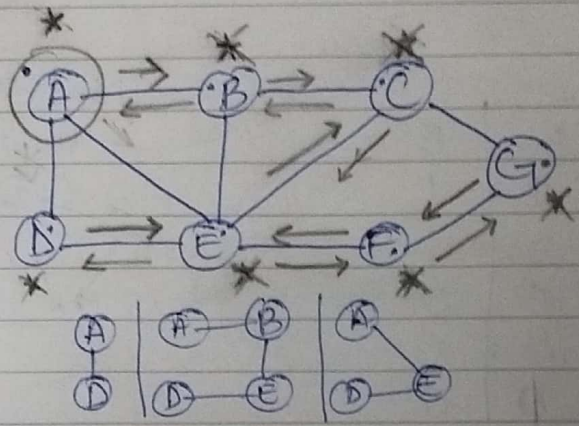
Graph Traversal
- Breadth First (BFS) Queue ↳ Stack.
- Depth First (DFS)

24/01/2023

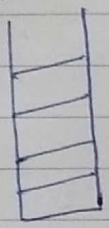
① Stack → Array, LL, Expression
② Queue → Array, LL, Implementation
③ Tree → LL, Array, BST, Insert, Search, Delete, Infix, Postfix, Prefix
④ Graph → LL, Array (2D) BFS, DFS, Djikstra.
⑤ Hashing → Linear, Quad, Double Hash.

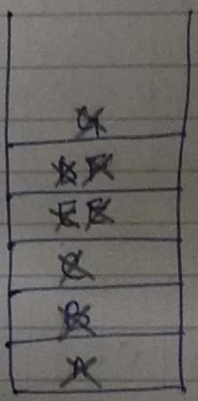
Balancing of Tree

BFS (Queue)
DFS (Stack).

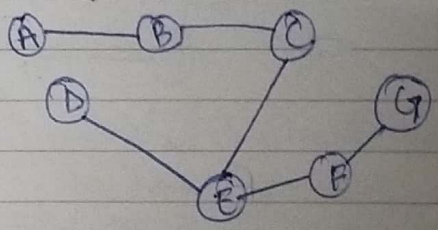


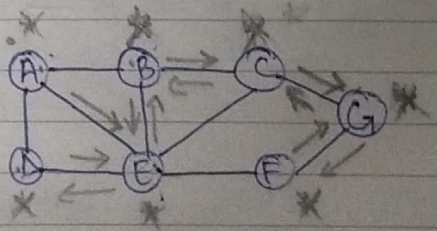
DFS

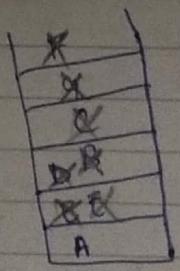




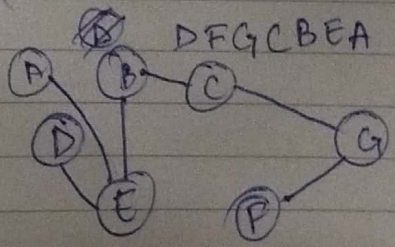
DGFECBA

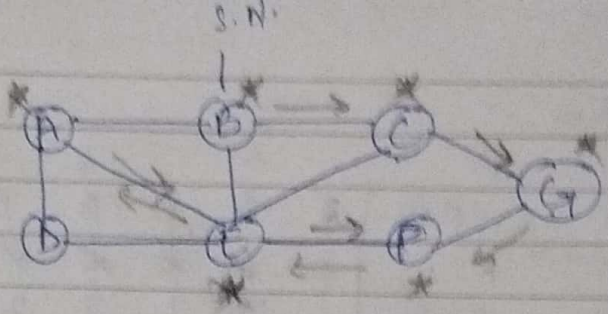


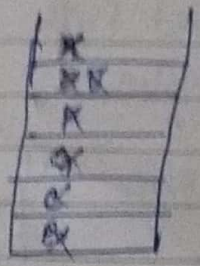




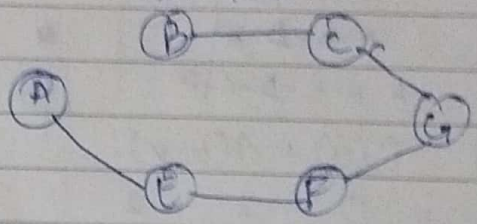
DFGCBEA



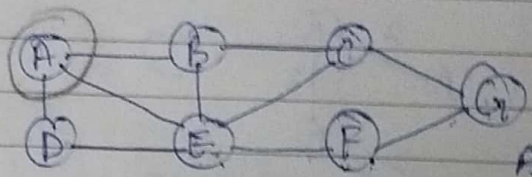




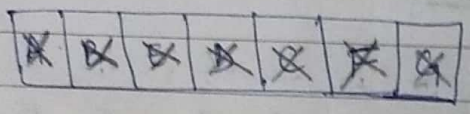
AEPGCB

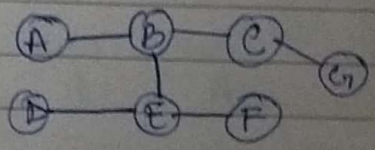


BFS

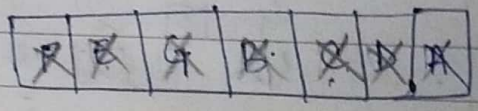


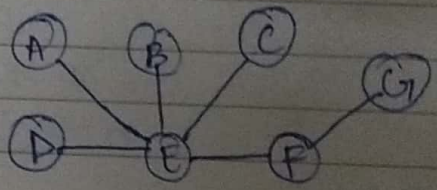
AEBCFG



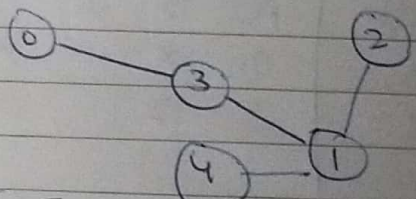


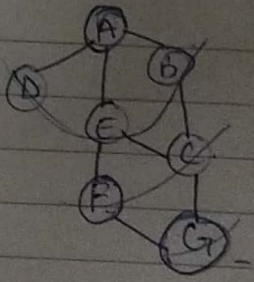
· ABEDCFG

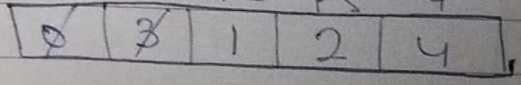




FEGBCDA



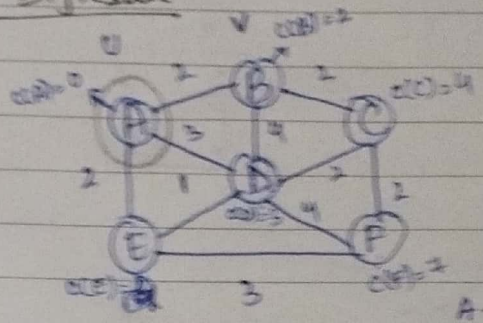




03124

Shortest Path Algorithm

Dijkstra

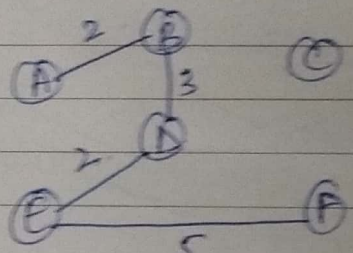


Path	A	B	C	D	E	F
A	0	~~∞~~	∞	∞	∞	∞
A-B		[2]		3	2	
~~A-B-D~~			4	[3]	2	
A-B-D-E			4	[2]		5
A-B-E-D-C-F			[4]			5

$C(v) = C(u) + d(u,v)$.

$= 0 + 2 = 2$

A - B - D - E - C - F



Src: A Dest: F

Do it again

Path	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
A-B		[2]		3	2	
A-B-D			4	[3]	2	
A-B-D-E			4		[2]	~~4~~
A-B-D-E-F						5

Path	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
A-E		2		3	[2]	
				3		5

31-01-23

HASH:

R ← L1 ← L2 ← RAM ← HDD

Hash Table → 1D

0	
1	11
2	32
3	
4	54
5	
6	
7	
8	
9	99

(K) → any key

(0 — m-1)

K % m = block #

h(K) = K % m

↓ hash function

Division Method

K	location/block h(K) = K % m
11	11 % 10 = 1
99	99 % 10 = 9
54	54 % 10 = 4
32	32 % 10 = 2

Multiplication method or Folding technique or Mid Square method.

Eg.

Collision.

Tags	Index	
13	1	} - collision.
14	1	
07	1	

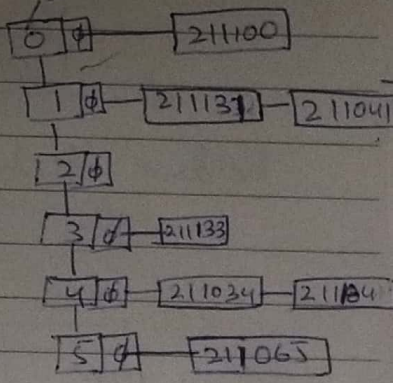
To resolve collision, there are two methods.

→ open hashing → no boundary to get solution.

→ close hashing → bounded to get solution

K	h(K)	Prob
211065	5	0
211100	0	0
211131	1	0
211133	3	0
211034	4	0
211017	7	0
211041	1	1
211184	4	
211147	7	

List of List Index.



open hashing / chaining

Close hashing

① Linear Probing
② Quadratic Probing

Probing = no. of collisions

Linear Probing.

$h'(K) = (h(K) + Prob) \% m.$

0	100
1	131
2	041
3	133
4	021
5	065
6	
7	
8	
9	

K	h(K)	Prob
065	5	0
100	0	0
131	1	0
133	3	0
034	4	0
017	7	0
041	1	01
184		
147		
021	1	XX 3

$h'(K) = (h(K)+P)\%m$
$= (1+1)\%10$
$2\%10 = 2$
$(1+3)\%10$
$4\%10 = 4$

Quadratic Probing

$h'(K) = (h(K) + Prob^2) \% 10.$

Mid square method.

(even) 2(10)65 (odd) 3(4)5

$10^2 \Rightarrow 100$ $4^2 = 16$

100 % m 16 % m

Folding technique

m = 10

2 10 65 3 45

21 03
10 +45
+65 48 % m
96

96 % m

j → indexes

k	hash 1	hash 2
065	(2*65+3)%10 = 3	(3*65+1)%10 = 6
100	(2*100+3)%10 = 3	(3*100+1)%10 = 1
131	(2*131+3)%10 = 5	(3*131+1)%10 = 2

$h(k) = h_1(k) + j\,h_2(k)$

$h_1(k) = (2k+3) \% 10$

$h_2(k) = (3k+1) \% 10$

0	1	2	3	4	5	6	7	8	9
			065	100	131				

$h(k) = h_1(k) + j\,h_2(k)$

$h(k) = 3 + 0*(6)$

$\boxed{h(k) = 3}$

$h(k) = 3 + 0 \times (1) = 3$ ✗

$h(k) = 3 + 1 \times (1) = 4$ ✓

$h(k) = 5 + 0 \times (2) = 5$ ✓

PW PAPERWORK