Time complexitivity

- no of steps required to solve the entire program

$$T(P) = C + t(P)$$

T(P) - Time complexity of program
C - Compile time
t(P) - Run time.

$$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \dots < 2^n < 3^n \dots < n^n$$

O big-oh : - upper bound.
Ω big-omega - Lower bound.
Θ theta - Avg bound

Big-oh

$F(n) = O(g(n))$ if $\exists$ +ve constants c and n_0 such that $F(n) \leq c * g(n)$ $\forall$ $n \geq n_0$

✱ Array

syntax-
type[] arrayname = new type[size]

ex:-
int [] arr = new int[size]
or
int arr[] = new int[3]

✱ int arr[] = {97, 98, 95};

✱ Stack [LIFO]

Real time ex-
- file
- books

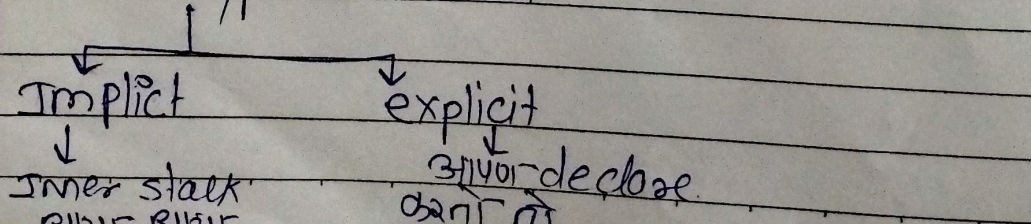
push → O(1)
pop → O(1)
peek → O(1) → means top

✱ Array - Fixed size so stack size is fixed.
- so we have to cheack for emply or full.

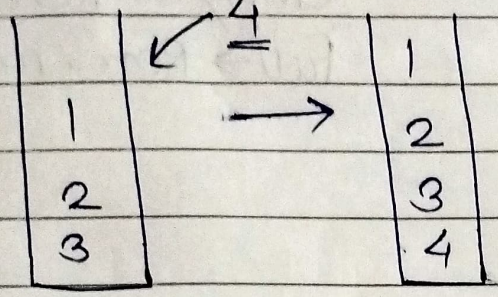
✱ In linkdlist stack is never full

Q push at the bottom of stack.
- possible using recursion

Two types of stack

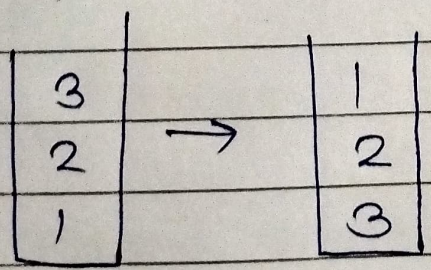


या आपण पहिले stack emply करू मग या ना new element टाकू मग परत सगळे element push करू.

ex 


```
void pushATbottom ( )
{
   PF (s. isempty())
   {
   s.push (data);
    return;
   }
int Top = s. pop();
pushATBottom (data, s);
   s.push (top);
}
```

② Reverse of stack.



या आपण new stack घेऊ मग सगळे टाकू

or

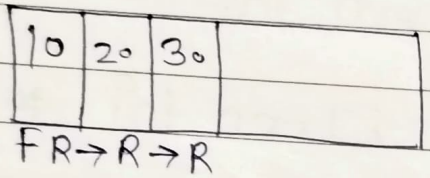
Recursion.

* Queue → FIFO

enque → Add
Deenque → Remove.
Front → peek

empty → Rear = = -1
full → Rear = max - 1

First Front = -1
Rear = -1



remove
{
int Front = arr[0]
for (int i=0; i<rear; i++)
{
arr[i] = arr[i+1];
}
rear--;
return Front;
}

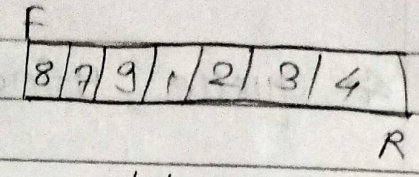
म्हणजे Front delete झाल्यानंतर बाकीचे पुढे shift करावा लागतात

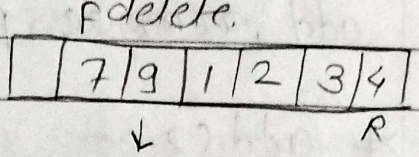
* add → O(1)
remove - O(n)
peek - O(n)

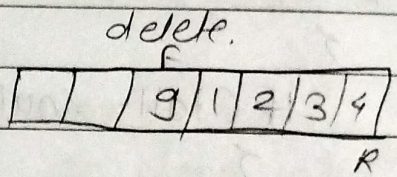
* Circular queue -

Using array

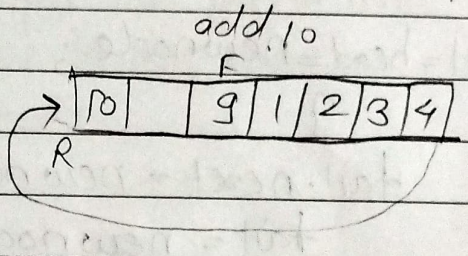
add
remove
peek
} O(1)







Full condtn
→ Rear+1 = front




```
isempty()
{
rear == -1 && Front == -1
}
```

```
isFull()
{
(rear+1) % size = front.
```

* Peek()
return arr[front]

```
* add()
{
  if (front == -1)
{
   front = 0;
}
rear = (rear+1) % size;
arr[rear] = data;
```

```
* remove
int result = arr[front]
if (rear == front)
{
  rear = front = -1;
}
else
{
  front = (front+1) % size
}
return result;
```

* queue using linked list.

add, remove, peek → O(1)

* add()

```
{
  IF(tail == null)
  {
tail = head = newnode;
  }
  tail.next = newnode;
    tail = newnode;
}
```

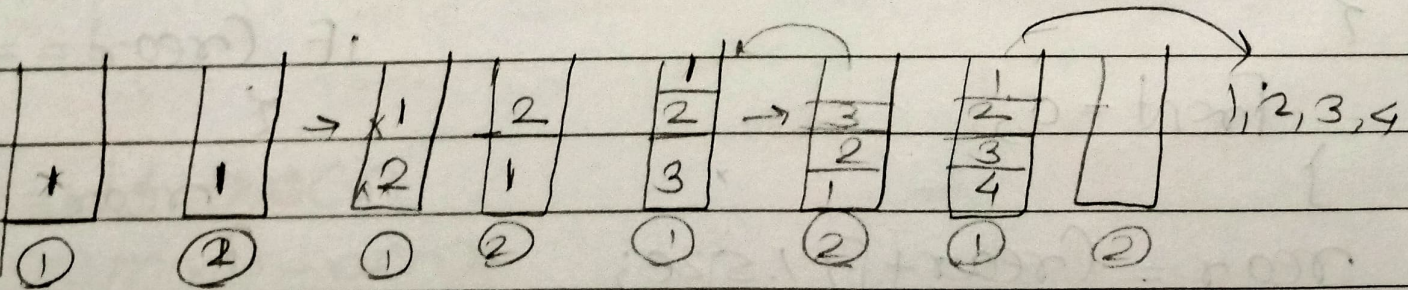
* remove()

```
{
int front = head.data;
iF(head == tail)
 {
   tail = null;
 }
head = head.next;
return front;
}
```

Q queue using 2 stacks

तर आपण पहिले एक element एक stack मध्ये टाकू मग तो stack दुसऱ्या stack मध्ये खाली करू and मग पुन्हा fill करू.

1, 2, 3, 4



* Appln of linked list in com

① Implementation of stack and queue.
② graph-
Adjacency matrix [to store adjacent vertices
③ dynamic memory allocation.
④ maintaing a directory of name.
⑤ performing arithmatic ope. on long int
⑥ polynomial store constant in the node.
⑦ representing sparse matrices.

Real world
- music player.
- Image viewer.

* Appln of circular LL

① We don't need to maintaine two pointers F
so We have front and rear.
② When multiple appln are running
③ circular doubly Linked List are used for the
implementation of advanced data struc like.
Fibonacci se Heap

* doubly LL Appln

- Redo and Undo functionality
- The most recently used section is represe
by dLL
- stack, Hashtable, binary tree.

Page No.:
Date:
YOUVA

* Array Vs LL
  - interview notebook.

* ArrayList vs LinkList

Insert	O(n)	>	O(1)
search	O(1)	<	O(n)

* Recursion :
  - The Funtn that calls itself

ex

$x = 2$

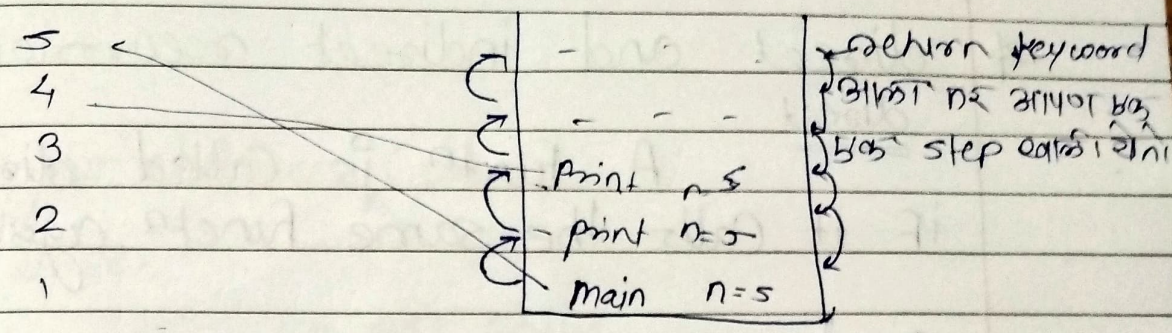
$F(x) = 2^2 = 4$

$F(F(x)) = F(4) = \underline{\underline{16}}$

outer function - हे consider करन की हे
inner function - सारख करत

ex :

```
print no(int n)
{
  if(n==0)
  return 0;
  syso(n);  → हा then - आणा - 5 print करले
  printno(n-1); → and ह्या - baki ch सारख
                  शेवटपर्यंत - 0 येत नाही
}                 output 5
n=x;
```



Recursion - need extra space for each n=value.
in loop we don't take new space for every ope.
but in recursion n takes always new space in memory.

* This tech of using is a complex program into simple program

* What is the base condtn in recursion.

- The condtn that is applied in any recursion functn is known as base condtn

जर base cond' दिली नसेल तर infinite time loop execute then stack overflow here.

* direct and indirect recursion

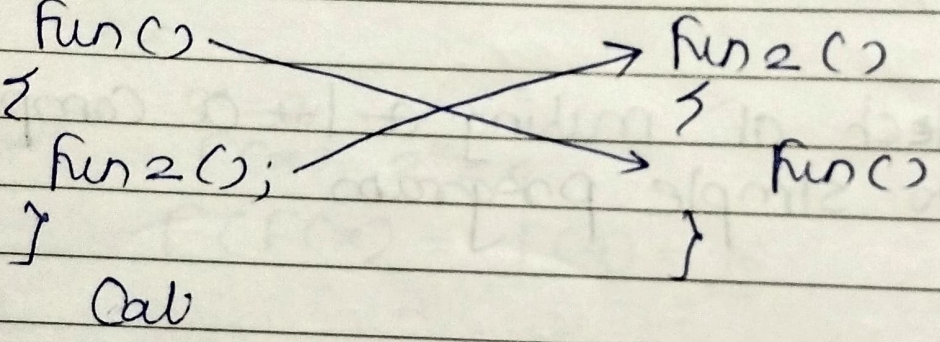
direct

A funtn is called direct recursion if it calls the same funetn again.

ex-

```
Fun()
{
  Fun()
}
```

Indirect Recursion



Call

* memory is allocated to different function calls in recursion.

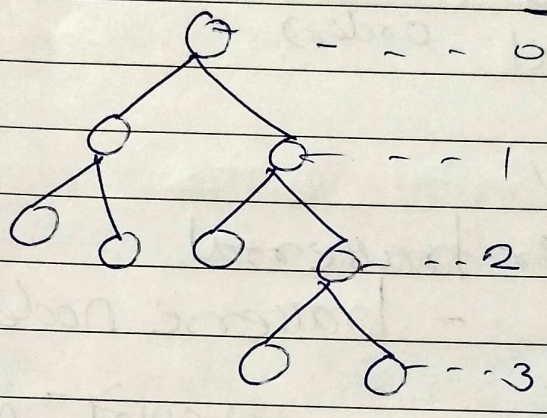
A recursive function calls itself so the memory for a called funetn is allocated on top of the memory [stack]

diff copy of local variables is created for each funtn call.

Trees

Leaf/ no children (last)
external/
terminal node.

internal node - एक वही child.



sibling - same parent.

Ancesters - पूर्वज
descenters - सगळे खालचे node एका.
node - each element of tree is node.

degree of node - no. of children to the node.

path - एखाद्या node पर्यंत येण्याचा मार्ग

ट्रावर्सल time complexity O(n)

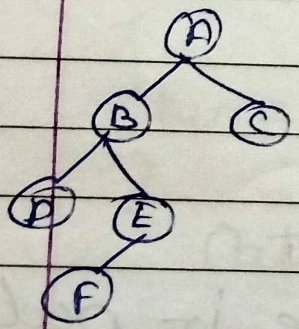
* Tree traversal :-

- Reading / processing the data of node exactly ones in some order.

- breadth First (level traversal order)
- depth first traversal.
  - pre
  - in
  - postorder

breath first / level order traversal

- traverse node in layers.



→ levelwise म्हणजे A नंतर म्हणजे Child B, C like...

A B C D E F

* depth

① pre VLR - A B D E F C
② in LVR - D B F E A C
③ post LRV - D F E B C A.

Level - 0   Depth - 0   Height - 0

* Binary trees

- can have max two children.

- At each level of i - max. no. of nodes → 2^i

- height → 3 → nodes → 4

- The max. no of nodes possible at height h is
  → h+1 → $n = h+1$

* if the no. of nodes is mini → height max
  ——————————————→ max → min

* max height → $\log_2 (n+1) - 1$

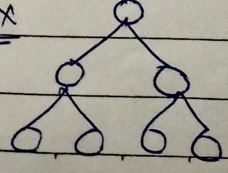
* preorder - VLR
  postorder - LRV
  Inorder - LVR

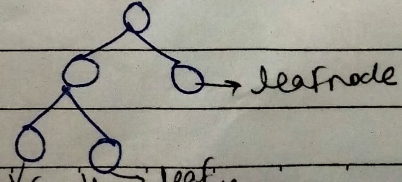
Inorder - data is shored.
postorder - root is at last.

* Complete binary tree :-

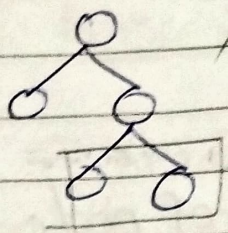
Binary tree in which every level except the last level is completely filled and all the nodes are left justified.

ex





जरा वरंच intermediate node रहावं filled झाले पाहिजे

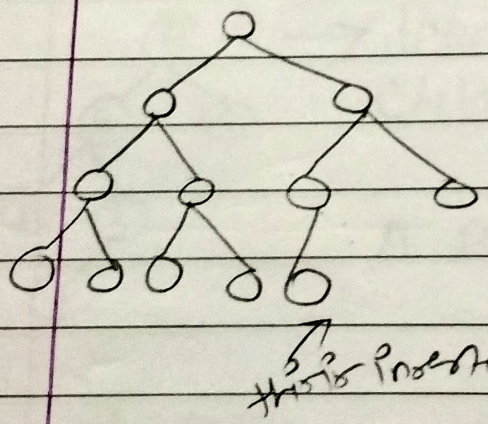


→ Right side में जहाँ left side में add करेंगे.

$$\text{depth} = \log_2(n+1)$$

* Almost Complete B.T →

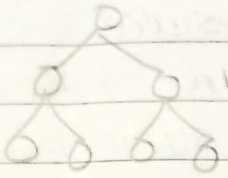
- superset is C.B.T

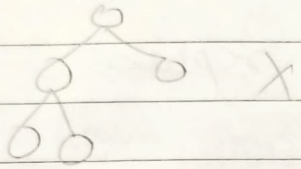


→ Here all intermediate node have have two children except one

this is inserted

* perfect binary tree -





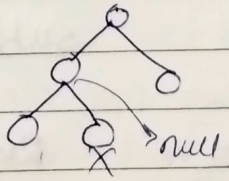
* Binary search tree - time complexicity - ?
  - at most 2 children.
  - The value of the left node must be smaller than the parent node.
  - The value of the right node must be equal or greater than root.

- searching is easy.

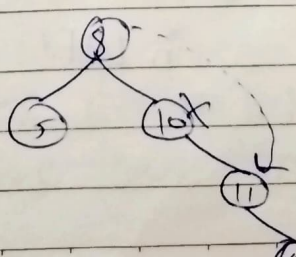
delete

① No child

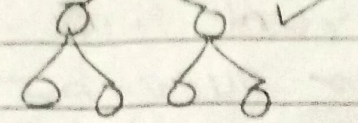
delete node & return null to parent

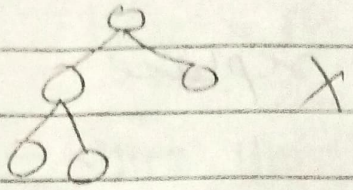


② one child.



delete node and replace with child node.





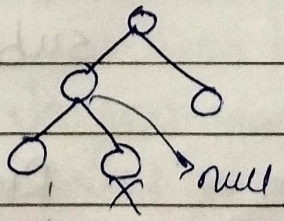
* Binary search tree - time complexicity - O
  - at most 2 children.
  - The value of the left node must be smaller than the parent node.
  - The value of the right node must be equal or greater than root.

- searching is easy.

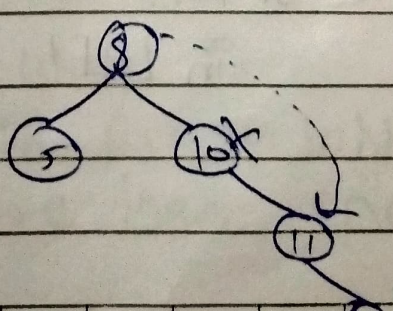
delete

① No child

delete node & return null to parent



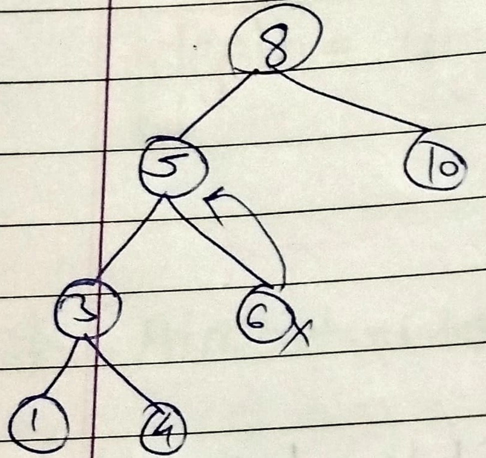
② one child.



delete node and replace with child node

③ two children

- Replace value with inorder successor.
- Delete the node for inorder successor



5 નેંતર 6 ને 5 replace by 5

જેના ના replace કરાયેં તેર left most in Right subtree.

* AVL tree is better than BST

- Balance factor(k) = height (left(k)) - height (right(k))

LL → Inserted node in left subtree of left subtree

RR - Right

* multi way tree

- m way tree is a BST with m child and m-1 keys.
-

child = m (no of child)
key = m-1

- left ka right value > left value.

* B-tree B+ tree.

B-tree	B+ tree.
① Data is stored in leaf as well as internal nodes.	① Data is stored in left nodes.
② searching is slower, deletion complex	② searching is faster, deletion easy.
③ No redundant search key present	③ present.
④ Leaf nodes not linked together.	④ linked together. ⑤ used to store the large amount of data which cannot be stored in ~~right~~ main memory

* B-tree

- balanced m-way (order) tree
- generalization of BST in which a node.
- can have more than one key and more than 2 children.

Key - root

- maintain sorted data
- all leaf node must be in same level.

* m

Every node has max m childen

min childen → leaf → 0
root → 2
internal nodes = $\lceil \frac{m}{2} \rceil$

key → (m-1)
min key → root → 1
other nodes → $\lceil \frac{m}{2} \rceil - 1$

- insertion always in leaf node

* For both

m=4

max childe = 4
min child = $\lceil \frac{m}{2} \rceil = 2$

max = 3

* B+ tree :-
- the size of main memory is always limited.
- internal nodes → stored
- leaf nodes → seeondary memory

- internal nodes called index nodes.

* Linear search algo / sequential

- Not found then returns null. (-1)
- It is used when unorderedlist.

	time	space
Best.	O(1)	→ O(1)
Avg	O(n)	
Worst	O(n)	

* Binary search
- divide and conquer approch
- only on sorted array

① Iterative method
② Recursive method.

Best	O(1)	space → O(1)
~~Worst~~ Avg	O(logn)	
Worst	O(logn)	

* collision - two keys same add

M T W T F S S
Page No.
Date
YOUVA

Remainder

① $H(K) = K \pmod m \rightarrow k \% m$

- choose m larger than no's
- To minimize collision, m should be prime or. no. with small divisors

* mid square method

$H(K) = u$.

- u is obtained by deleting the digits from both ends of K^2

ex key 10 → 10^2 → 100 → 0 position.

$15^2 \rightarrow 225 \rightarrow$ 2 position.

* Folding method

$H(K) = K_1 + K_2 + K_3 + \ldots + K_n$

keys are partitioned into parts

- parts are added together.

key → 10

K_1 K_2

1 + 0 → 1 index

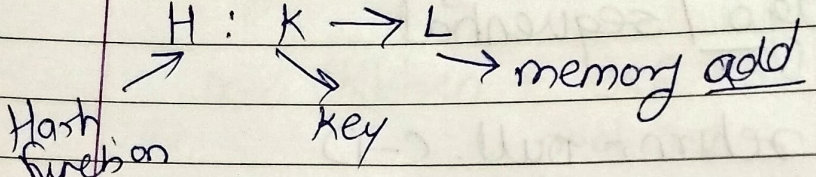
2 1

2 + 1 → 3 → index

* Hashing - constant time

- insert and retrieve data in O(1) time.
- indexing
- mapping

to insert We use hash function



* Drawback of Hash Funct^n

- Hash function assigns each value with a unique key
  - same key generates so collision

Hash Funct^n - We can calculate the add. at which the value can be stored

Types

① division method

Remainder

① $H(K) = K \pmod m \rightarrow K \% m$

- Choose m larger than no's
- To minimize collision, m should be primeor. no. with small diviisors

* Mid square method

$H(K) = l$.

- l is obtained by deleting the digits from bo ends of K^2

ex key $\rightarrow 10 \rightarrow 10^2 \rightarrow 100$ 0 position.

$15^2 \rightarrow 225$ 2 position

* Folding method

$H(K) = K_1 + K_2 + K_3 + \ldots + K_n$

keys are partitioned into parts

- parts are added together.

key $\rightarrow 10$

$K_1 \; K_2$

$1 + 0 \rightarrow 1$ index

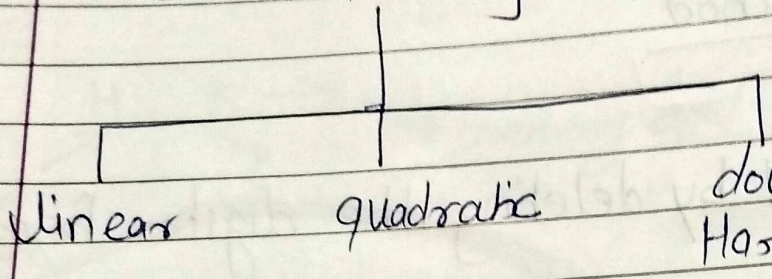
2 1

$2 + 1 \rightarrow 3 \rightarrow$ index

✱ Collision resolution

① ~~Linear~~

open Hashing (closed addressing)
↳ chaining
closed Hashing (open addressing)



① Chaining

0	10	13
1	12	15
2		
3		

(2k+3) → formula
2k+3 % 10
Lets say
13 → 0

So linked list is used.

✱ linear

0	13	7	
1	9	8	
2		9	3
3			
4			
5	6		
6	11		

nearly
4T
n

key	location	probes
3	9	1
2	7	1
9	1	
6	5	
11	5 → 2	
13	9 → 2	
7	7	

12 7

Hash function → $h(k,i) = (h'(k) + i) \bmod m$

$i = 0, 1, 2$

② quadratic

→ $(a+i)^2 \% m$ $i = 0$ to $(m-1)$

0	
1	9
2	
3	3
4	
5	6
6	11
7	2
8	
9	13

key	location	probe
3	9	1
2	7	1
9	1	1
6	5	1
11	5	2
13	9	2

location

(11) → $5 + (0) \% 10 = 5$ start from 0

$5 + 1 \% 10 = 6$ ✓

(13) $9 + 0 \% 10 = 9$

$9 + 1 \% 10 = 0$ ✓

③ double Hashing

→ $(u + v * i) \% m$ $i = 0$ to $(m-1)$

$v = [h_2(k) \% m]$

$h_2 = 2k + 3$

0
1
2
3
4
5
6
7
8
9

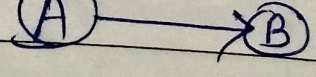
6

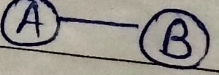
Key	Location(u)	v	probe
6	$[2\times6+3] \% 10 = 5$	-	
11	$[2\times11+3] \% 10 = 5$	4	
	$= 3$		1 term

★ Graph

- ordered set G(V, E)
  - V(G) - set of vertices
  - E(G) - set of edges.

Directed graph - with direction



Undirected graph → 

path- seq. of nodes that are followed in order to reach some terminal node V from initial node U.

closed path- If the initial node is same as terminal node. $V_0 = V_N$

simple path- If all the nodes of the graph are distict with an exception $V_0 = V_N$

cycle- some path exists betn every two vertices. There is no isolated nodes.

Complete graph-
every node is connected to all other nodes.
$\frac{n(n-1)}{2}$ edges. $n \rightarrow$ vertex

Weighted graph-
Weight assign.

Digraph- each node edge of graph is associate with some direction
-and traversing can be done only in the specified direction.

loop- similar end points

Adjacent / neighbours
- edge is connected then

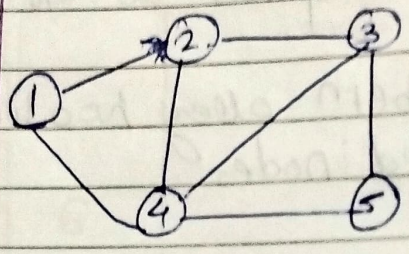
Degree of the node.

- no. of edges of that vertex

Isolated vertex have 0 degree

* Adjacency matrix

→ dense graph



	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	1	0
3	0	1	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0

Space Complexity O(n²)

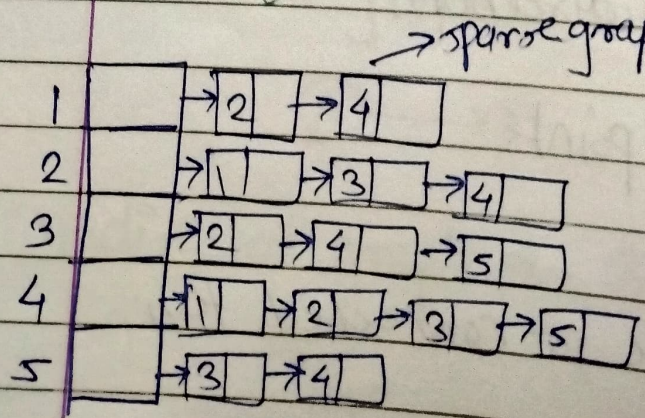
Undirected graph so mirror img

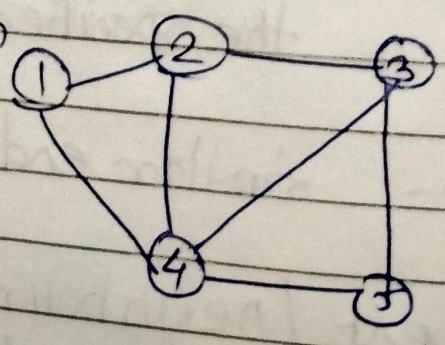
It is also called as sequential representation.

Weighted graph में Weight in matrix

* Adjancy list / LikedList

→ sparse graph





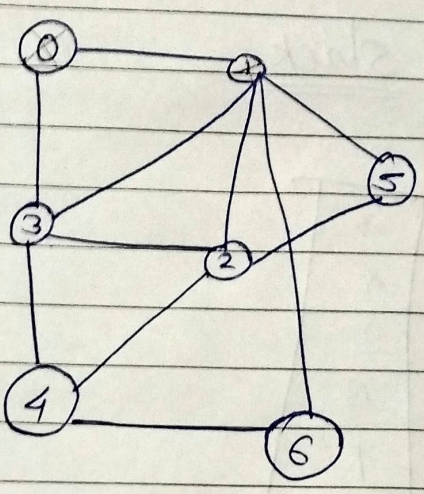
Sp

Space Com → O(n+2e)

* BFS → Breath first search (level order)

Queue

0	1	3	2	5	6	4



first zero enter then delete that zero. and insert the nearest. edge vertices direct connection.

Result → 0, 1, 3, 2, 5, 6, 4

- It is a recursive algo.
- peer - to peer network,
- BitTorrent, uTorrent. to find seeds and peers.

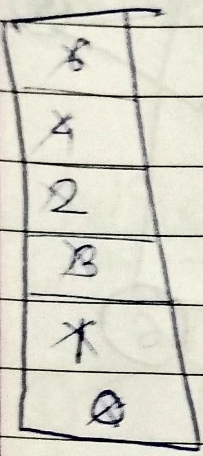
- Used in Web crawlers to create Web pages.
- Used to determine the shortest path and minimum spanning tree.
- Cheney's technique to duplicate the garbage collection
- Used in Ford - Fulkerson method to compute the max flow in the flow network.

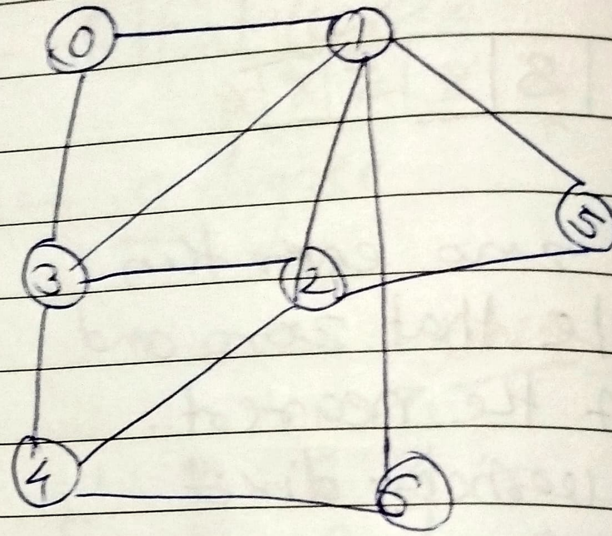
Time complexicity →
best → O(V+E)
worst → O(V)

Space → O(V).

DFS → depth first search

Stack





Result → 0 1 3 2 4 6 5

any node adjacent to 0

now, 6 પછી નંબર 5 ના ભાઈના કોઈ નહિ છે.
આપણે 6 છોડીને ગયા 4 ને try નહિ ગયા 2 ને

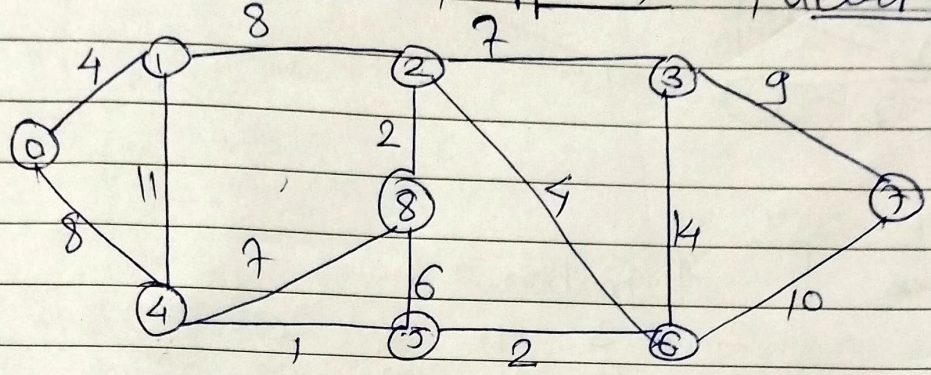
- recursion

appln - topological sorting
- to detect cycles
- one soln puzzles
- graph is bipartite or not

Shortest path

☆ Dijkstra's algo.

- single-source shortest
  greedy approch, level-sitting



BIRT $0 \to 1 = 4$ $\quad$ $① \to ② \to 8+4 = 12$

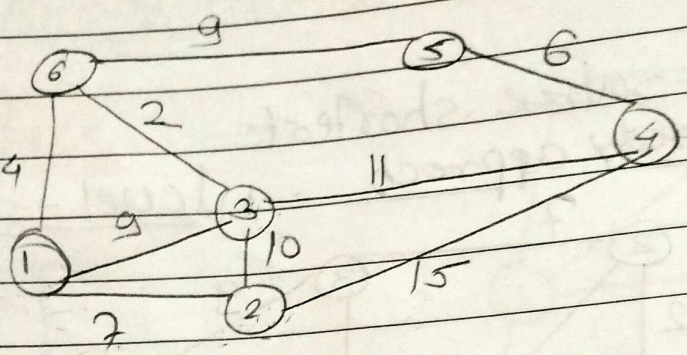
$0 \to 8 = 8$ $\quad$ $① \to ④ \to 11+4 = 15$

IF $d(u) + c(u,v) < d(v)$

$d(v) = d(u) + c(u,v)$

0	1	2	3	4	5	6	7	8
0,1	(4)	∞	∞	8	∞	∞	∞	∞
0,1,2	(4)	(8)	∞	8	∞	∞	∞	∞
0,1,2,6	(4)	(8)	7	8	∞	(4)	∞	14
	(4)	(8)	7	8				

0	1	2	3	4	5	6	7	8
0,1	(4)	∞	∞	8	∞	∞	∞	∞
0,1,4	(4)	12	∞	(8)	∞	∞	∞	∞
0,1,4,5	(4)	12	19	(8)	(9)	∞	∞	15
0,1,4,5,6		12	19			(11)	∞	15
0,1,4,5,6,2		(12)	20	25			21	15



Source	destination				
	2	3	4	5	6
1	∞	∞	∞	∞	∞
1,2	7	9	∞	∞	14
1,2,3	7	9	22	∞	14
1,2,3,6	7	9	20	∞	11
			1→3→4	20 1→3→6→5	1→3→6

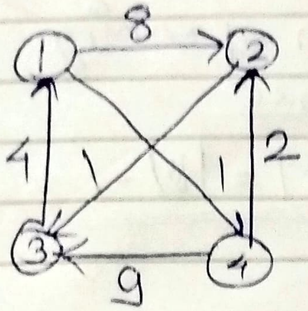
20

20 20 सो कोणतीही more

1,2,3,6,4,5 → 20

येथे आता जर 0 ने start केले तर smallest find out मग नंतर तो add but every line 0 पासून ते पण select चाले आहेत ते घेऊन calculate करत येत no necessary to select all and Calculate Distance.

Floyd-Warshall algo.



D^0 =

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

direct distance

D^1 =

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	3	0

1 → column and row both copy as it is.

पहिले 1 vertex मध्ये घेऊन बघा मग 2 vertex घेऊन 3.....

last D^4 =

	1	2	3	4
1	0	3	4	1
2	5	0	1	6
3	4	7	0	5
4	7	2	3	0

Here, first we find the shortest distance means direct edge betn two points then take 1 vertex betn the two then calculate the smallest distance if the distance is min then replace the distance then take 2 vertex in middle

This is same as the dijkstra's algo same in process.

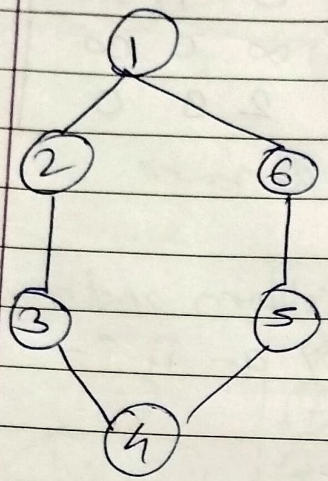
Minimum spanning tree

greedy approach

① Prims algo

In spanning tree

$|E| = |V| - 1$

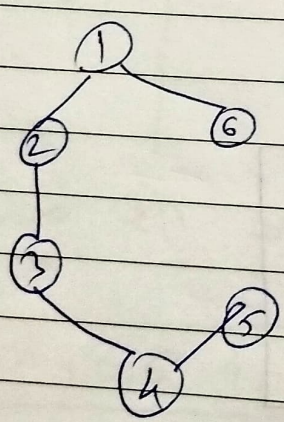


spanning tree

How many spanning tree

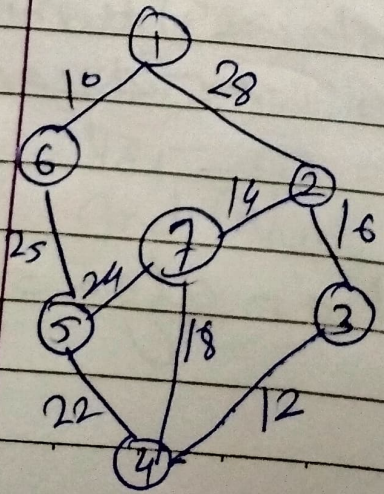
$|E|_{C_{|V+1|}}$ – no. of cycles

closed path form that thing.

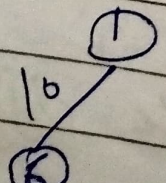


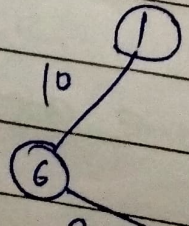
just every vertex should be connected to someone, i.e called spanning tree.

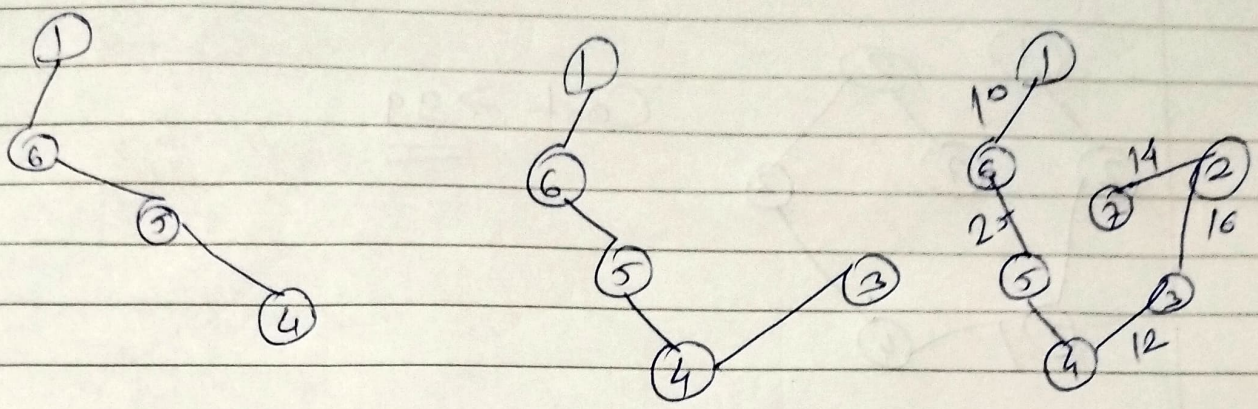
Prims



Select first edge with min cost

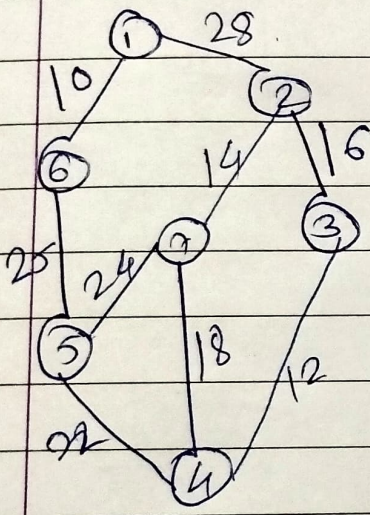




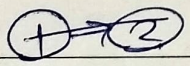


Cost = 99

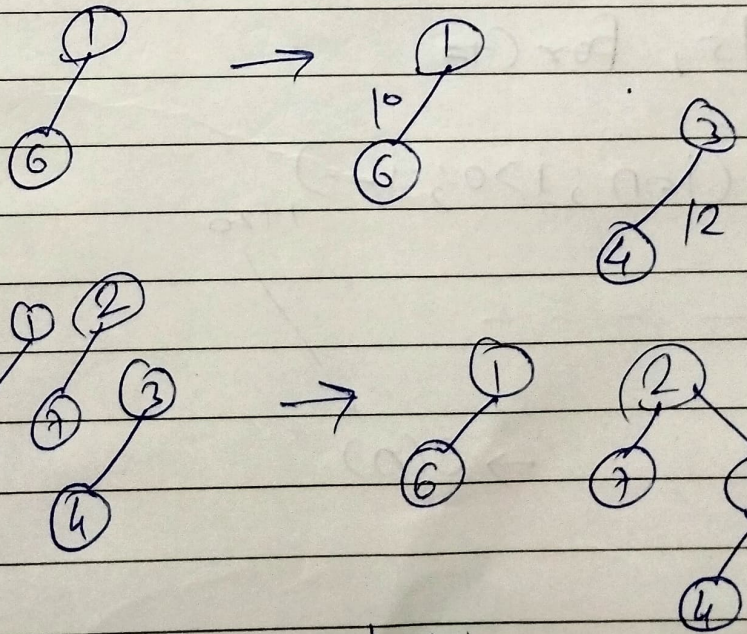
② Kruskal's algo



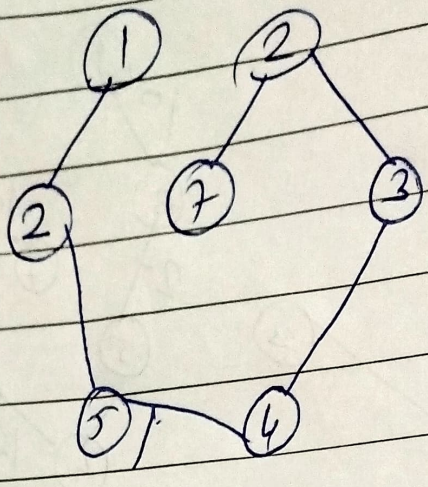
- select the min edge which do not form a cycle.



smallest path.



but if is means 7→4 then cycle is created so don't take that edge



Cost → 99

MinHeap → $O(n \log n)$

✱ Time Complexicity

① For (i=0 ; i<n ; i++) → n+1
↳ i+2, i/2

{
--- → n
}

→ O(n)

Lets, For (i=

For (i=n ; i>0 ; i--) i+20

{

}

→ O(n)

```
for (i=0 ; i<n ; i++)
{
  for (j=0 ; j<n ; j++)
    {
      - - -
    }
}
```

→ $O(n^2)$

*
```
p=0
for (i=1 ; p<=n ; i++)
{
    p = p+i;
}
```

Here we don't know

<u>so</u> if $p>n$ then break so.

$O(\sqrt{n})$

*
```
for (i=1 ; i<n ; i = i*2)
{
}
```