18/01/23

Week-1

1. Problems on Stack:-

* Next Greater Frequency Element

Aim:- Given an array, for each element find the value of the nearest element to the right which is having a frequency greater than that of the current element. If there does not exist an answer for a position, then make the value '-1'.

Example:-

Input : a[] = [1, 1, 2, 3, 4, 2, 1]

output : [-1, -1, 1, 2, 2, 1, -1]

Explanation:-

Given array a[] = [1, 1, 2, 3, 4, 2, 1]

Frequency of each element is: 3, 3, 2, 1, 1, 2, 3

Let calls Next Greater Frequency element as NGF

1. For element a[0] = 1 which has a frequency = 3, As it has frequency of 3 and no other next element has frequency more than 3 so '-1'

2. For element a[1] = 1 it will be -1 same logic like a[0]

3. For element a[2] = 2 which has frequency = 2, NGF element is 1 at position = 6 with frequency of 3 > 2

4. For element a[3] = 3 which has frequency = 1 NGF element is 2 at position = 5 with frequency 2 > 1

5. For element a[4] = 4 which has frequency = 1, NGF element is 2 at position = 5 with frequency 2 > 1

6. For element a[5] = 2 which has frequency = 2, NGF element is 1 at position = 6 with frequency of 3 > 2

7. For element a[6] = 1 there is no element to its right hence -1

Input: a[] = [1, 1, 1, 2, 2, 2, 2, 11, 3, 3]

output: [2, 2, 2, -1, -1, -1, -1, 3, -1, -1]

* Steps :-

1. Create a list to use values as index to store frequency of each element.

2. push the position of first element to stack.

3. pick rest of the position of elements one by one and follow steps in loop.

   i) Mark the position of current element as 'i'.

   2) If the frequency of the element which is pointed by the top of stack is greater than frequency of current element, push current position i to the stack.

   3) If the freq of element which is pointed by the top of stack is less than freq of current element and stack is empty. then

      i. Continue popping the stack

      ii. If the condition in step c fails then push the current position i to stack.

4. After the loop in step 3 is over, pop all the elements from stack and print -1 as next ngf element for them does not exist.

2. * How to Reverse a String using Stack

Aim:- Given a string, reverse it using stack.

Example:-

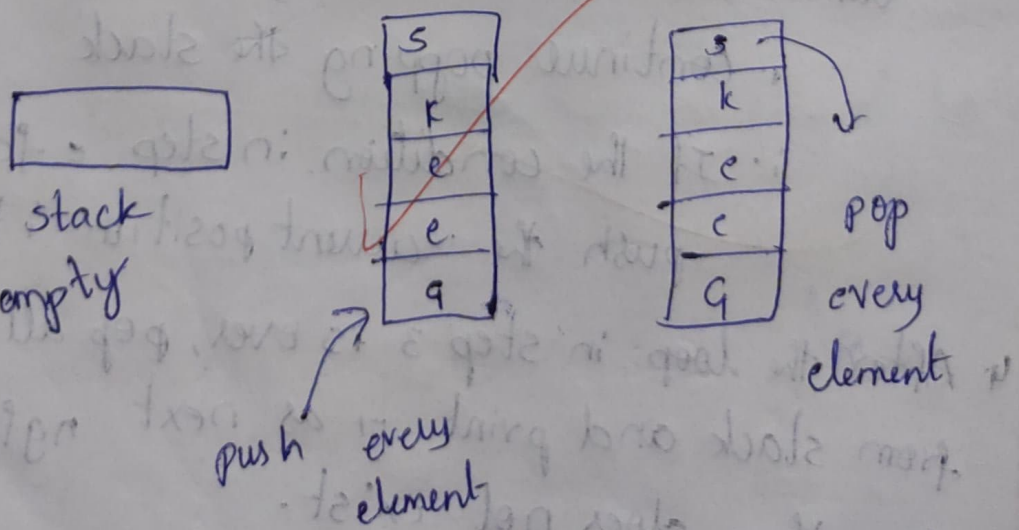
Input:- str = "GeeksQuiz"

output:- ziuQskeeG

Explanation:-

For reverse a string using stack

* Create an empty stack
* one by one push all characters of string to stack
* one by one pop all characters from stack and put them back to string.



25/01/23

Week-2

1. problems on Queue:-

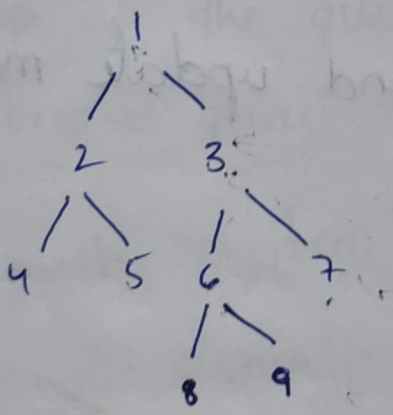
* print Right View of a Binary Tree

Aim:- Given a Binary Tree, print the Right view of it.

The right view of a Binary Tree is a set of nodes visible when the tree is visited from the Right side.

Example:-

Input:-



output:- Right view of the tree is 1 3 7 9

Explanation:-

The idea is to use recursion and keep track of the maximum level also. And traverse the tree in a manner that the right sub-tree is visited before the left subtree.

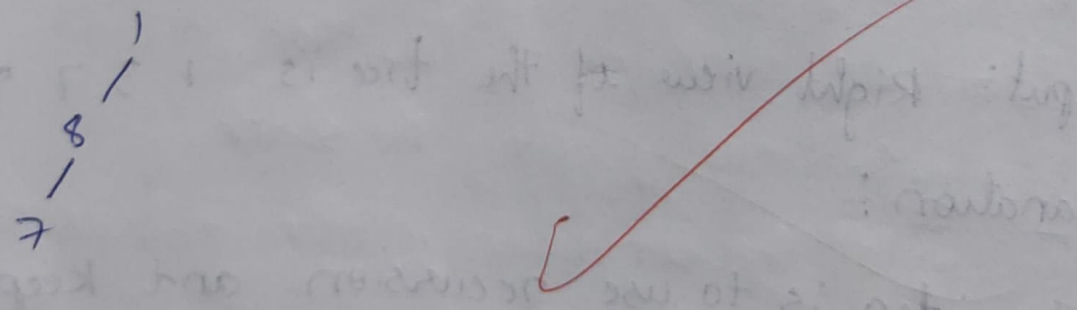
Steps:-

* perform postorder traversal to get the rightmost node first.

* Maintain a variable name max_level which will store till which it prints the right view

* While traversing the tree in a post-order manner if the current level is greater than max_level then print the current node and update max level by the current level.

Input:-



output:

Right view of tree is 1 8 7

2. Implementing a stack using the queue data structure

Aim:- Design a stack that supports push and pop operations using standard enqueue and dequeue operations of the queue.

Explanation:-

To implement the queue's enqueue operation such that the last entered item always ends up at the queue's front. we need additional queue.

1. To push an item into the stack, first move all elements from the first queue to the second queue, then enqueue the new item into the first queue, and finally move all elements back to the first queue. This ensures that the new item lies in front of the queue and hence would be the first one to be removed.

2. To pop an item from the stack, return the front item from the first queue.

output:

5

4

3

.

.

Underflow !!

1/02/23

Week-3

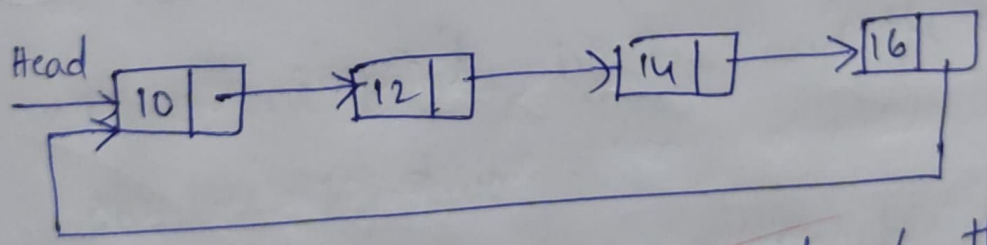
1. * problems on linked list:-

Aim:- check if a linked list is circular Linked list

* Given a singly linked list, find if the linked list is circular or not.

Explanation:-

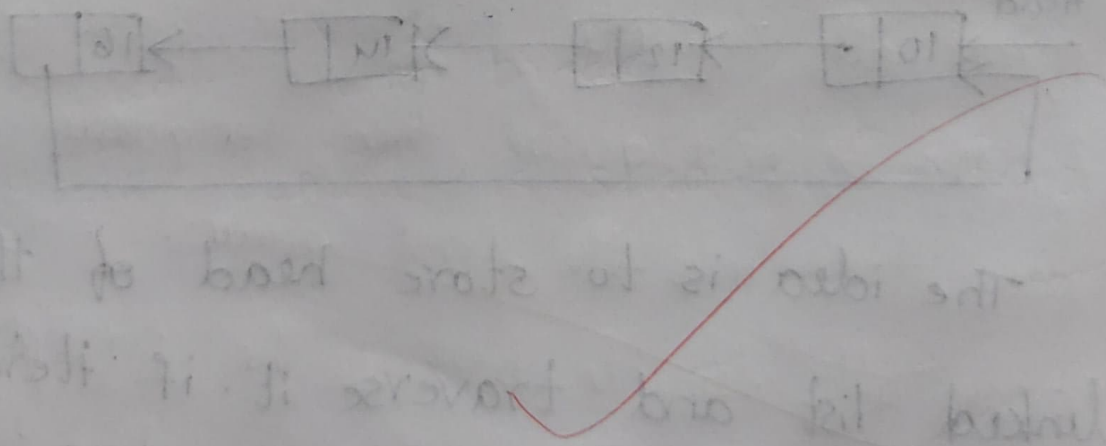
A linked list is called circular if it is not NULL-terminated and all nodes are connected in the form of cycle.



The idea is to store head of the linked list and traverse it. if iterator reaches NULL, linked list is not circular. else if it reaches head again, it is circular.

Steps :-

* Declare a Node pointer and initialize it to the head's next.
* Move node pointer to the next node, while the node is not equal to nullptr and node is not equal to the head.
* After coming out of the loop, check if the node is equal to head then return true, else return false



2. Aim:- Find the middle element of a given linked list

~~Aim~~ Given a single linked list, find the middle of linked list.

Example:-

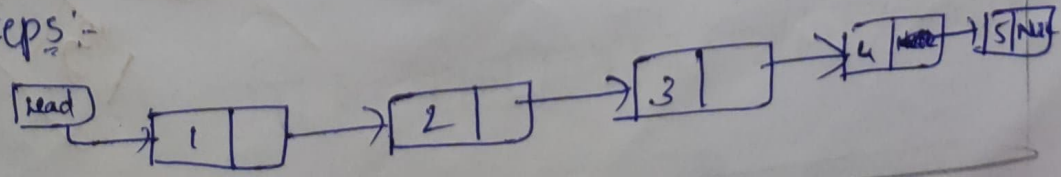
$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$

output:-

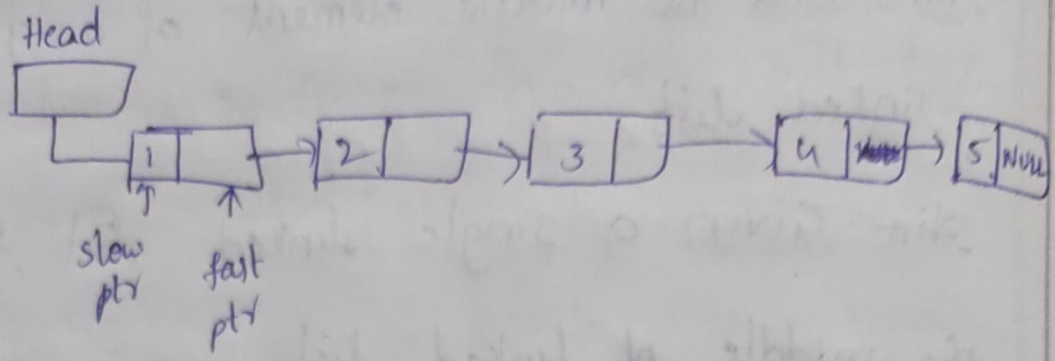
3

* Enter element in the linked list
* Traverse linked list using two pointers.
* Move one pointer by one and the other pointer by two.
* When the fast pointer reaches the end, the slow pointer will reach the middle of the linked list.

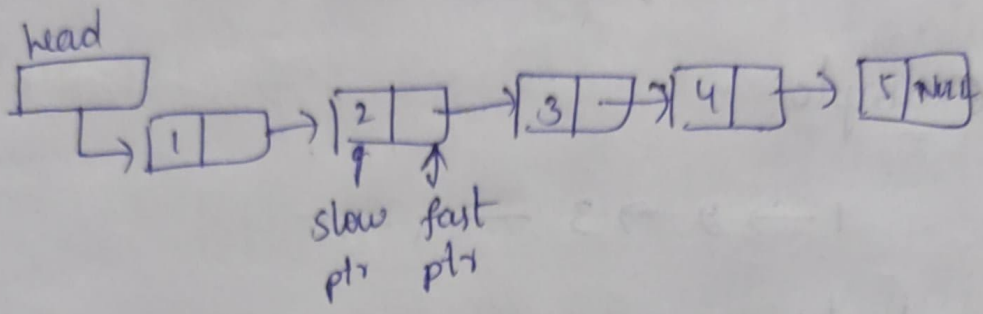
Steps:-

1) 

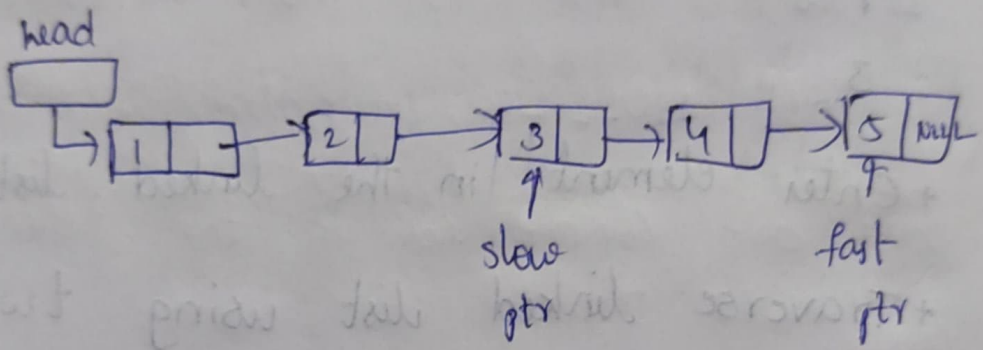
2)



3)



4)



Here fast->ptr -> Next = NULL

So, while loop break

Middle element is slow->ptr -> Data

18/2/23

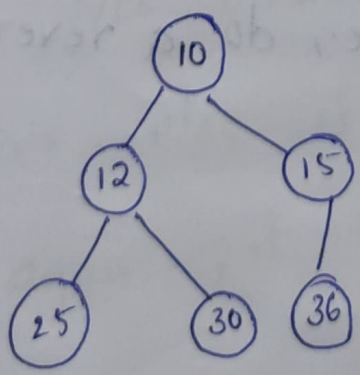
1. problems on Double Linked list

Aim:- To convert a given Binary Tree to double linked list in linear time.

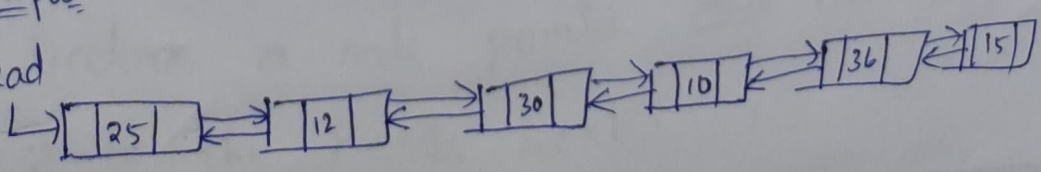
Explanation:-

The left and right pointers in nodes are to be used as previous & next pointers respectively in converted DLL. The order of nodes in DLL must be same as Inorder of the given Binary Tree. The first node of Inorder traversal must be head node of the DLL.

example:-



output:-

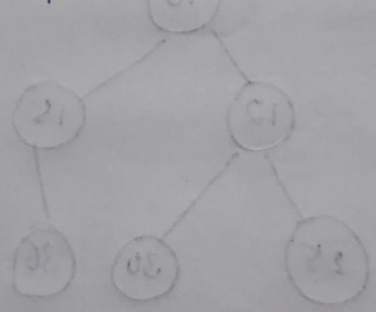


Steps :-

* Below are the three different solutions have been discussed for this problem.

First method :-

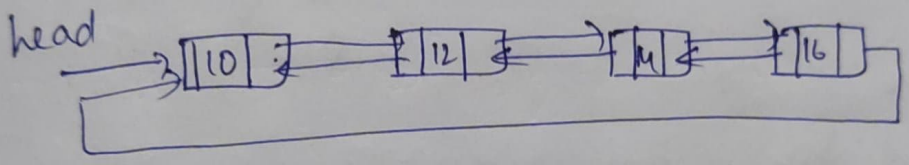
* In the following implementation, we traverse the tree in inorder fashion. we add nodes at the beginning of current linked list and update head of the list using pointer to head pointer. Since we get insert at the beginning. we need to process leaves in reverse order. for reverse order, we first traverse the right sub-tree before the left sub-tree i.e., do a reverse inorder traversal.



2. Aim:- check if a linked list is a circular linket list.

Explanation:-

A linked list is called circular if it is not null terminated and all nodes are connected in the form of the cycle.



This idea is to store head of the linked list and traverse it. if iterator reaches NULL, linked list is not circular. else if it reaches head again, it is circular.

Steps:-

1. Declare a node pointer and initialize it to the heads next.

2. move node pointer to the next node. while the node is not equal to null ptr & node is not equal to head.

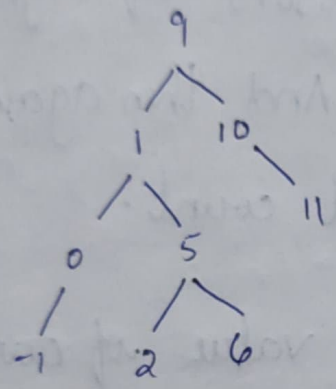
3. After coming out of the loop, check if the node is equal to head then return true, else return false.

*problems on Trees

Aim:- To count the greater nodes in AVL tree.

Example:

x=5



output:- There are four values which are greater than 5 in AVL tree. they are 6, 9, 10, and 11.

Explanation:-

→ first we maintain an extra field "desc" for storing the no. of descendant nodes for every node.

→ for calculating the number of nodes which are greater than given value we simply traverse the tree. while traversing three

cases can occur.

1. Case - x (Given value) is greater than the value of the current node.

2. case - x is lesser than the value of current node. we increase the current count by no. of successors of right child of current node. And then again add two to the current count.

3. case-x is equal to the value of current node. In this case we add the value of desc field of right child of current count & then add one to it.

2. Aim:- To find the minimum no. of nodes in an AVL tree with given height.

→ Given the height of an AVL tree 'h', the task is to find the minimum no. of nodes the tree can have.

Example:-

Input : H = 3

output : N = 7

→ there are two approaches to find the minimum no. of nodes.

1. Recursive Approach:-

In an AVL tree, we have to maintain the height balanced property. ie; diff in the height of the left & right subtrees can not be other than -1, 0 & 1 - for each node.

Steps:-

* for height = 0, we can only have a single node in an AVL tree, ie. n(0) = 1
* for height = 1, we can have a minimum of two nodes in an AVL tree, i.e. n(1) = 2

* Now for any height 'h', root will have two subtrees. out of which one has to be of height (h-1) & other (h-2)

* So, $\boxed{n(h) = 1 + n(h-1) + n(h+2)}$ is the required recurrence relation for $h >= 2$

2. Tail Recursive Approach:-

* The recursive function for finding n(h) is $n(h) = 1 + n(h-1) + n(h-2)$;

$h >= 2$; $n(0) = 1$; $n(1) = 2$;

* To create a tail recursive function, we will maintain $1 + n(h-1) + n(h-2)$ as functions arguments such that rather than calculating it, we directly returns its value to main function.

Week-6

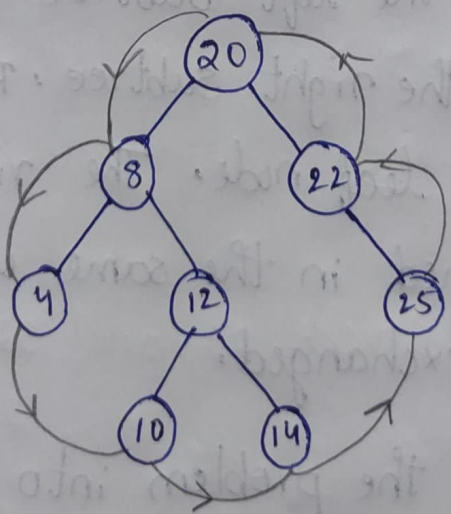
1. Aim:- Boundary Traversal of Binary Tree.

Given a binary Tree, print boundary nodes of the binary tree Anti-clockwise starting from the root.

The boundary includes

1. Left boundary (nodes on left excluding leaf nodes)
2. leaves (consist of only the leaf nodes)
3. right boundary (nodes on right excluding leaf nodes)

Example:- boundary traversal of tree is



This is how we write the traversal

root : 20 , left-boundary nodes : 8

leaf nodes : 4 10 14 25

right-boundary nodes : 22

Description:-

The left boundary is defined as the path from the root to the left-most node. The right boundary is defined as the path from root to the right-most node. If the root doesn't have left subtree or right subtree, then the root itself is left or right boundary.

The left-most node is defined as a leaf node you could reach when you always firstly travel to the left subtree if it exists. if not, travel to the right subtree. Repeat until you reach a leaf node. The right-most node is also defined in the same way with left and right exchanged.

Approach:- Break the problem into 3 parts

1. Print the left boundary in top-down manner.

2. print all leaf nodes from left to right, which can again be sub-divided into two sub-parts :

2.1 print all leaf nodes of left sub-tree from left to right

2.2 Print all leaf nodes of right sub-tree from left to right

3. Print the right boundary in bottom-up manner.

Note :- Nodes are not printed again. e.g. The left most node is also the leaf node of the tree

Output :-

20 8 4 10 14 25 22

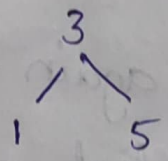
2. Aim :- Merge two BSTs with limited extra space

Given two Binary search Trees (BST), print the inorder traversal of merged BSTs.

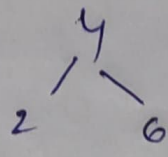
Example :-

Input:-

First BST



Second BST



output:- 1 2 3 4 5 6

* Steps to solve :-

1. Consider two stacks s1 and s2 which stores the elements of the two trees.
2. store the left view value of a tree1 in s1 and of tree2 in s2.

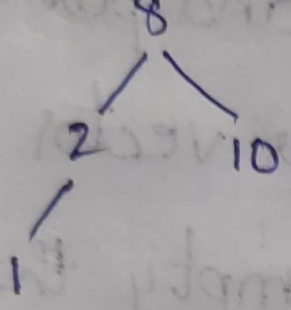
3. Compare the top values present in the stack and push the value accordingly in the result vector.

* if s2 is empty then pop s1 and put the popped node value in the answer vector
* Else if both s1 & s2 are not empty then compare their top nodes value if s1.top()→val <= s2.top()→val then in this case push the s1.top()→val in the result vector and push its right child in the stack s1.
* If s1 is empty then pop s2 and put the popped node value in the answer vector.
* Else if both s1 and s2 are not empty then compare their top nodes value if s2.top()→val >= s1.top()→val then in this case push the s2.top()→val in the result vector & push its right child in the stack s2.
* Loop while there are nodes not yet printed. The nodes may be in the stack (explored, but

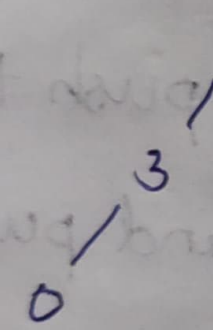
not printed) or maybe not yet explored

Input:

First BST



Second BST:



output:- 0 1 2 3 5 8 10

Week - 7

1. Aim :- Find the number of islands using DFS.

Given a binary 2D matrix, find the number of islands. A group of connected 1s forms an island.

For example, the below matrix contains 5 islands.

Example :-

Input: mat[][] = {{1, 1, 0, 0, 0},
{0, 1, 0, 0, 1},
{1, 0, 0, 1, 1},
{0, 0, 0, 0, 0},
{1, 0, 1, 0, 0}}

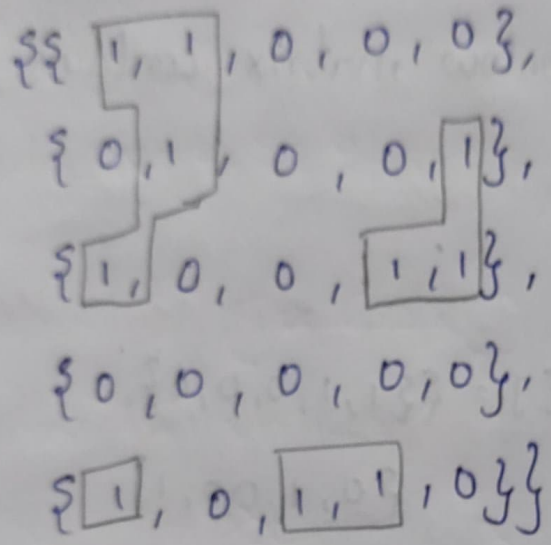
output :- 5

A graph where all vertices are connected with each other has exactly one connected component, consisting of the whole graph. Such a graph with only one connected component is called a strongly connected graph.

what is an island?

A group of connected 1s forms an island.

For example, the below matrix contains 4 islands



* Steps to solve :-

* Initialize a boolean matrix visited of the size of the given matrix to false.
* Initialize count=0, to store the answer.
* Traverse a loop from 0 till ROW
  * Traverse a nested loop from 0 to COL
  * if the value of the current cell in the given matrix is 1 and is not visited
  * call DFS function

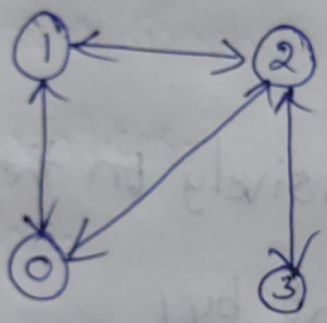
* Initialize rowNbr[] = {-1, -1, -1, 0, 0, 1, 1, 1} and colNbr[] = {-1, 0, 1, -1, 1, -1, 0, 1} for the neighbour cells
* Mark the current cell as visited
* Run a loop from 0 till 8 to traverse the neighbor
* if the neighbor is safe to visit and is not visited
* call DFS recursively on the neighbor.
* Increment count by 1
* Return count as the final answer

2. Aim:- Detect cycle in undirected graph

Given an undirected graph, the task is to check if there is a cycle in the given graph.

Example:-

Input:- N=4, E=4



output:- Yes

Explanation:- The diagram clearly shows a cycle 0 to 2 to 1 to 0

* Steps to follow :-

* Iterate over all the nodes of the graph and keep a visited array visited[] to track the visited nodes.

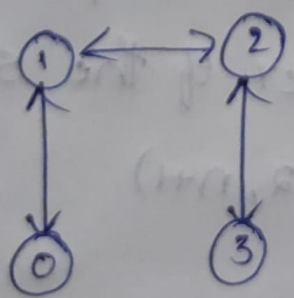
* Run a Depth First Traversal on the given

subgraph connected to the current node and pass the parent of the current node.

In each recursive

* set visited [root] as 1.
* Iterate over all adjacent nodes of the current node in the adjancency list
* If it is not visited then run DFS on that node and return true if it returns true.
* Else if the adjacent node is visited and not the parent of the current node then return true.
* Return false.

Input:- N = 4, E = 3, 0, 1, 2, 1, 2, 3



output:- No

Week-8

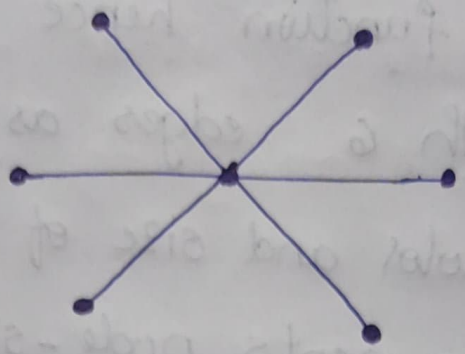
1. **Aim:-** Star Graph using Networkx Python

A star graph is a special type of graph in which n-1 vertices have degree 1 and a single vertex have degree n-1. This looks like that n-1 vertex is connected to a single central vertex. A star graph with total n-vertex is termed as S_n.

Properties of star Graph :-

* It has n+1 vertices.
* It has n edges.
* It does not have any cycle.
* The diameter of a star graph S_n is a minimum of (2,n).
* A star graph is a tree.
* It has no unconnected component.
* The chromatic number of the star graph is a minimum of (2, n+1)

Example of S_6:



Approach :-

* We will import the required networkx module
* After that, we will initialize a number of nodes to 6.
* we will create graph object G using star_graph() function.
* We will realize the graph using nx.draw() function.
* We will make the color of nodes green and increasing size by passing extra arguments to nx.draw().

Explanation:-

As we passed 6 as an argument to the star-graph() function hence we got a star graph with 6 edges as output. We changed the color and size of nodes by passing extra arguments node-size and node-color to the nx.draw() function.

2. Aim:- Create a cycle graph using Networkx in Python

A cycle graph is a graph which contains a single cycle in which all nodes are structurally equivalent therefore starting and ending nodes cannot be identified.

Properties:-

* No. of nodes in a cycle graph (C_n) are equal to N.
* No. of edges in a cycle graph (C_n) are equal to N.
* Every Node is connected to 2 edges hence degree of each node is 2.
* It is a Hamiltonian cycle graph.
* It is a connected graph.
* Even after removal of one edge it remains connected as it is cyclic.
* We can simply obtain cycle graph by joining the initial an final node of a path graph.

Module Used:-

Networkx module for realizing a cycle graph. This module in python is used for visualizing and analyzing different kind of graphs.

It contains in built function

networkx.cycle_grap() and can be illustrated using the networkx.draw() method.

Approach:-

* import module
* Create cycle graph object using cycle_graph() constructor
* Use nx.draw() function
* Display plot

output :-