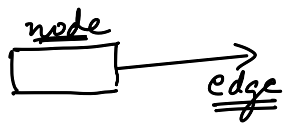
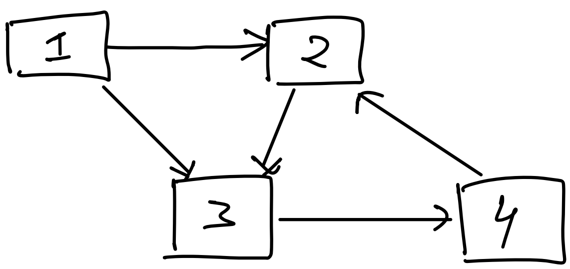
Welcome ☺

Agenda : Graphs
Terms
Cycle detection
2-3 questions.

Graphs

⇒ Graph is a collections of nodes & edges

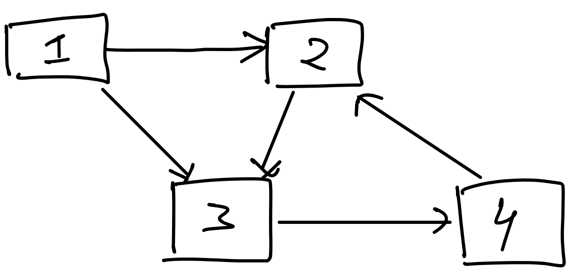




nodes = 4
edges = 5

How graphs are stored.

1) Adjacency Matrix

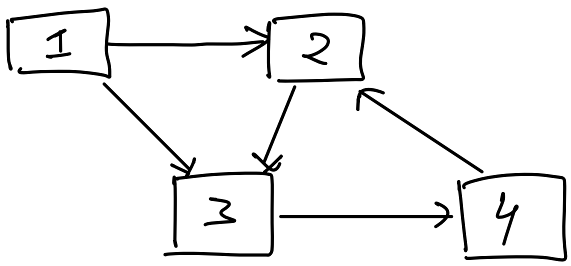


	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	1
4	0	1	0	0

S.C ⇒ $O(N^2)$

$A[i][j] \to$ 1, $\boxed{i} \to \boxed{j}$
$\to$ 0, no edge.

2. Adjacency List



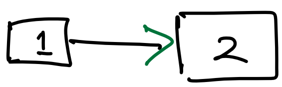
1 → { 2, 3 }
2 → { 3 }
3 → { 4 }
4 → { 2 }

list < list < > >
array < list < > >

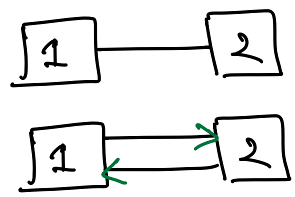
SC ⇒ $O(N+E)$

Properties of Graph

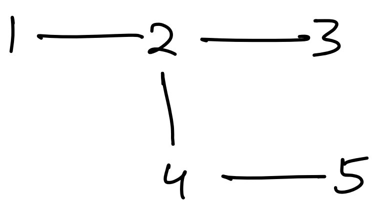
1. Directed



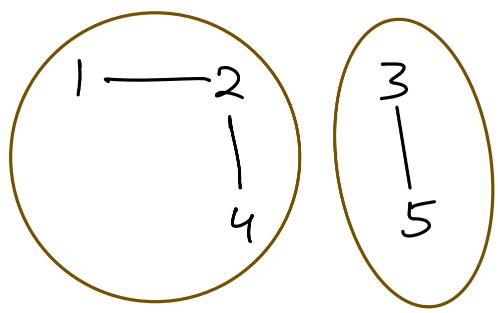
Undirected



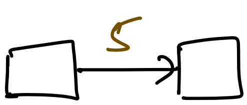
2. Connected



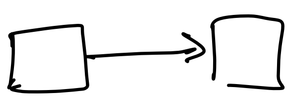
Disconnected.



3. Weighted



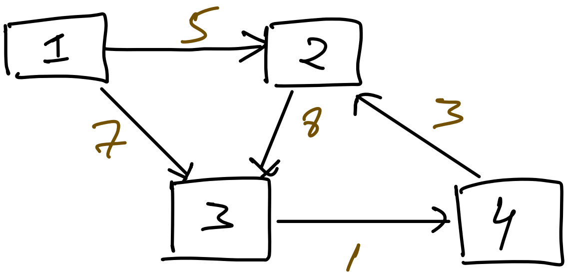
Unweighted.



$A[i][j] \to 5\ (>0)$ $\quad i \xrightarrow{5} j$

$\quad\quad\ \ \to 0$, no edge

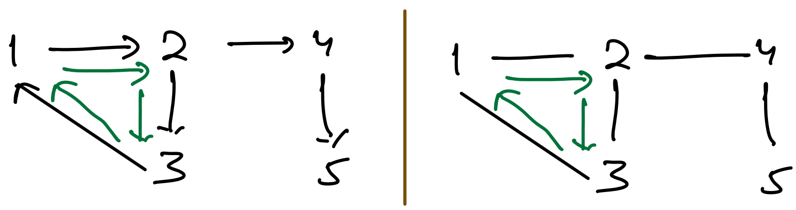
$Adj[i] \to$ list of pairs (j, wt)

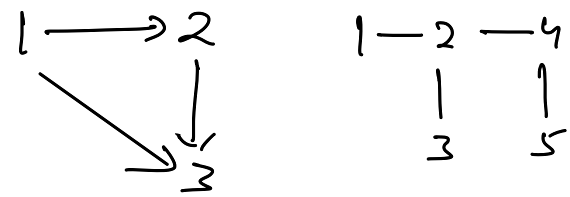


$1 \to \{(2,5), (3,7)\}$
$2 \to \{(3,8)\}$
$3 \to \{(4,1)\}$
$4 \to \{(2,3)\}$

4:

Cyclic | Acylic

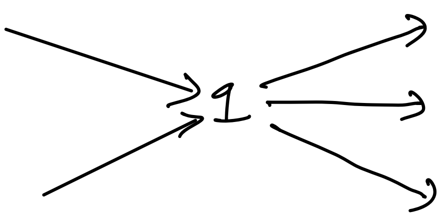




$\Rightarrow$ undirected graphs $\to$ cycle of min. 3 nodes will be considered.

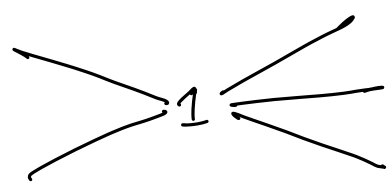
5:

Indegree / Outdegree.



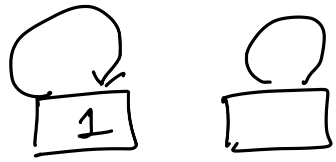
in[1] $\Rightarrow$ 2
out[1] $\Rightarrow$ 3

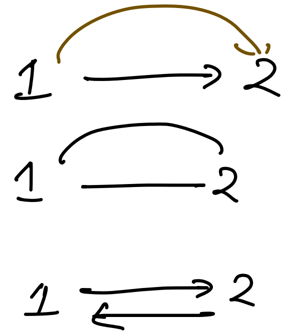
Degree



degree[1] = 5

6: Simple Graph $\Rightarrow$ connected graph without self edge & multiedges



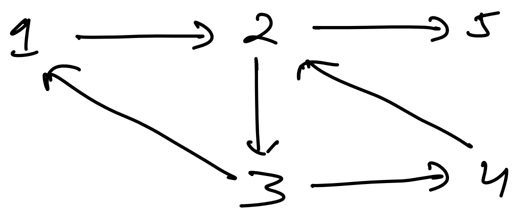


X not multiedge

Traversal

Depth First Search

→ Go depth first till it is possible. Once the path is completed, backtrack and try alternate paths.



Code

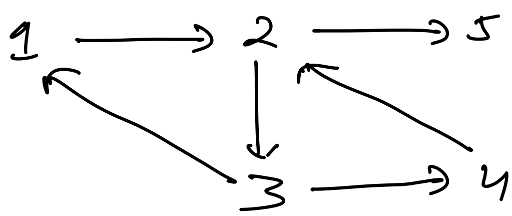
```
∀i: vst[i] = false

for ( i -> 1 to N)
{
   if ( ! vst[i]) dfs[i]
}

void dfs (u)
{
   vst[u] = true
   print (u)
   for ( y : adj[u] )
   {
      if ( ! vst[y]) dfs(y)
   }
}
```

To ensure we don't miss traversing a node.

S.C $\Rightarrow O(N+N)$

T.C $\Rightarrow O(N+E)$

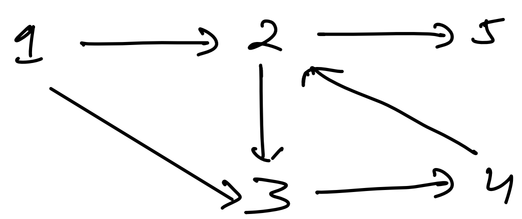
1. Travel all nodes only once.
2. Keep track of visited nodes
3. Check if all nodes are travelled before exit.



T	T	T	T	T
1	2	3	4	5

1 2 3 4 5

Q Check if given graph has cycle ?



1 → 2 → 3 → 4 → 2

If a visited node is travelled again ⇒ cycle ✗

If a visited node in same path is travelled again ⇒ cycle ✓

travel a path ⇒ DFS

Code

```
∀i vst[i] = false
for( i → 1 to N)
{ if( !vst[i] && dfs[i])   return true
}

bool dfs ( u)
{
  vst[u] = true
  path[u] = true
  for( y : adj[u] )
  {
     if( path[y] == true)   return True.
     if( !vst[y] ) {
         if( dfs(y) )   return true
     }
  }
  path[u] = false
  return false
}
```

T.C ⇒ O(N + E)

SC ⇒ O(N + N + N)

Q You are given a 2D grid '1' (land) and '0' (water). Your task is to determine # islands in the grid. Diagonal connection not allowed.

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

ans = 3

	1	2	3	4
1	0	~~1~~ 2	~~1~~ 2	0
2	0	0	~~1~~ 2	0
3	0	0	0	1
4	0	1	0	0

Code

```
islands = 0
for( i -> 0 to N-1 )
{
    for ( j -> 0 to N-1 )
    {
        if ( graph[i][j] == 1 )
        {
            islands ++
            dfs( i, j )
        }
    }
}
```

```
              L   T   R   D
row       [ 0 , -1 , 0 , 1 ]
column    [ -1 , 0 , 1 , 0 ]
```

```
void dfs( i, j )
{
    graph[i][j] = 2
    // move to neighbours.
    for ( k = 0 ; k < 4 ; k++ )
    {
        x = i + row[k]
        y = j + col[k]
        if ( x >= 0 && x < N && y >= 0 && y < N
                && graph[x][y] == 1 )
        {
            dfs( x, y )
        }
    }
```

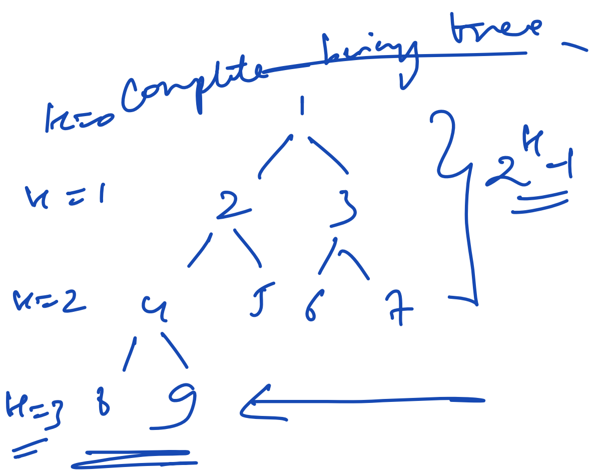
Q Linkedin maximise reach.

H.W

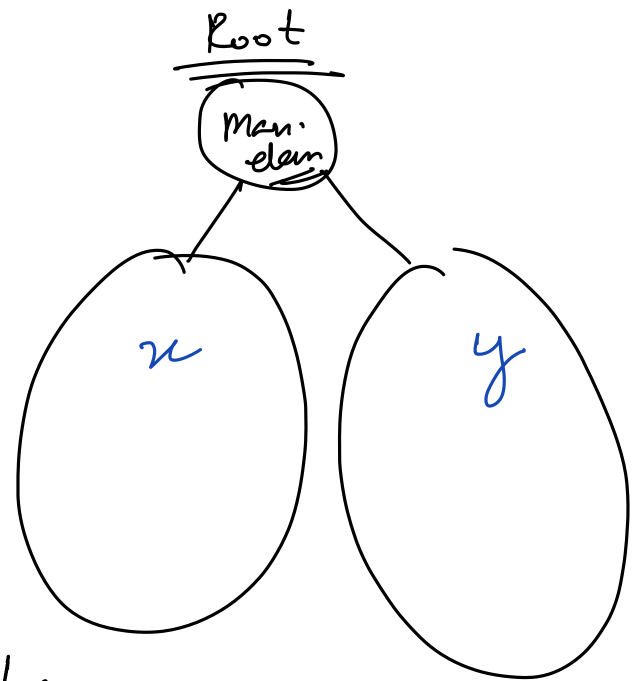
Doubt

Distinct max heaps. → complete binary tree.

distinct numbers



$H=3 \Rightarrow 2^3-1 = 7$



$^{N-1}C_x$

$^{N-1}C_y$

getDistinct(___ , N)

x y

$^{N-1}C_x$ * getDistinct(x) * getDistinct(y)

$$\frac{2^H - 1 - 1}{2}$$

$\left.\begin{array}{l}\text{\# of nodes in left subtree} \\ \text{\# of nodes in \underline{right} subtree}\end{array}\right\}$ except last level

$N - (2^H - 1)$