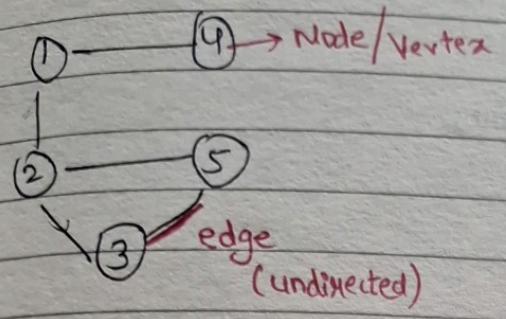
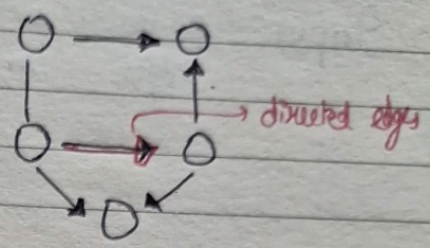
GRAPHS

Raj
DATE / /
PAGE

Undirected Graph.



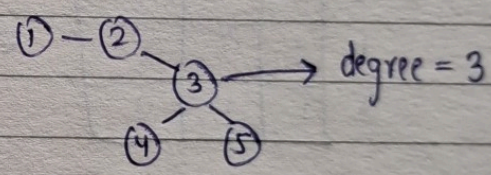
Directed graph



CYCLES : If you traverse from a node and end up on that node it mean's graph cycle

PATH : Contains nodes and each of them are reachable.

eg: 1 2 3 5 → Path ✓
1 2 3 1 → not Path

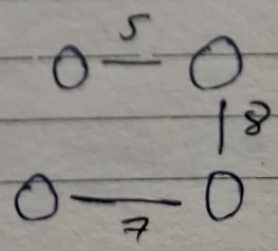


DEGREE: No. of edges that go in and out of that node

- Total degree of graph is twice of no. of edges

1) IN-DEGREE : No of incoming edges
2) OUT-DEGREE : No of outgoing edges

EDGE WEIGHT : Default wt = 1



weight of edges (random)

Graph Representation

Input : nodes : n
edges : m

n	m
2	1
1	3
2	4
3	4
2	5
4	5

m lines

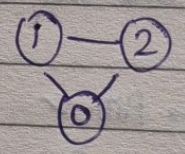
I : UNDIRECTED GRAPH

① Matrix representation

adj matrix [n+1] [n+1] {where n = no. of elements (nodes)

	0	1	2
0	0	1	1
1	1	0	0
2	1	1	0

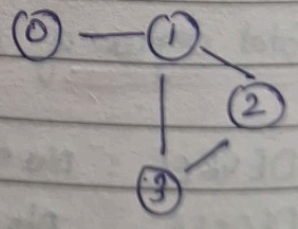
⇌



② Adjacency List representation (BETTER)

Space = O(E)

0 → 1
1 → 0, 2, 3
2 → 1, 3
3 → 1, 2



Adj list:

CODE

```
vector<int> adj[n+1]
cin>>n>>m;
for(i=0 to m)                    // Input "m" lines
{ int u,v
  cin>>u>>v;
  [ adj[u].push_back(v);
    adj[v].push_back(u);
  ]
}
```

only this in directed graph ← [adj[u].push_back(v);]

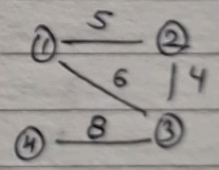
] Push for Both nodes, as in Undirected graph. Both nodes Share same edge.

II DIRECTED GRAPH

Store same as in Adjacency list, except in code

CODE all same except

```
{ adj[u].push_back(v); }
  ← no second push →
```



• WEIGHTED GRAPH

Matrix

	0	1	2	3
0	0	5	6	0
1	5	0	4	0
2	6	4	0	8
3	0	0	8	0

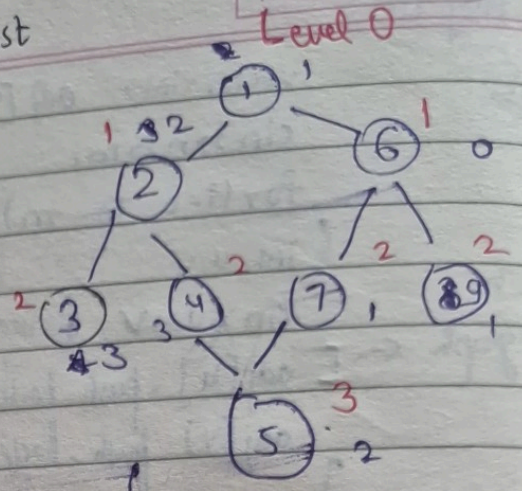
Adj. List (Store Pairs)

1 → (2,5)
2 → (3,4) (1,5)
3 → (4,3) (1,6) (2,4)
4 → (3,8)

Pair (node, wt)

BFS for Adj list

- We select a starting Point acc^n to that node, (Level 0)
- We traverse level wise (determined) (by distance from start node
- Traversal change acc^n to starting point



BFS : [1, 2, 6, 3, 4, 7, 8, 5]

for starting point = 1

BFS : [6, 1, 7, 8, 2, 5, 3, 4]

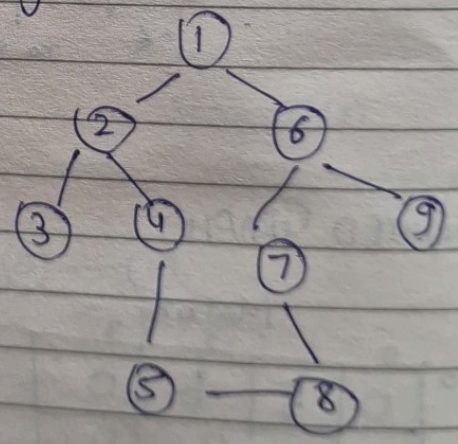
for start Point = 6

1) Make a queue and a visited arr to check if element is visited or not

we have the Adj list.

0 = { }
1 = { 2, 6 }
2 = { 1, 3, 4 }
3 = { 2 }
4 = { 2, 5 }
5 = { 4, 8 }
6 = { 1, 7, 9 }
8 = { 5, 7 }
9 = { 6 }

eg:



N = no. of Nodes (0 based Index)

its like matrix {array of vector}

```
vector<int> BFS( int N, vector Adj[] )
{
    int vis[N] = {0};
    vis[0]=1; // put vis[0]=1 as we insert first element
queue<int> q
    q.push(0);
    vector<int> bfs;

while ( !q.empty() )
{
    int node = q.front();
    q.pop();

    bfs.push_back(node); push that value taken from queue

    // Now push its members if they are not visited
    for ( auto it : adj[node] )
    {
        if (!vis[it])
        {
            q.push(it);
            vis[it]=1;
        }
    }
}

return bfs;
}
```

STEPS

Take each index of list, add to queue

Pop from Queue, visit all its members and add to queue.

just before adding in queue, always check if element is visited or not

Space comp: O(3N)

Time comp: O(N) + O(E)

Adj list — vertices — edges

Adj matrix TC = O(N²) + O(E)

DFS for Adj List

we traverse recursively in depth

Code

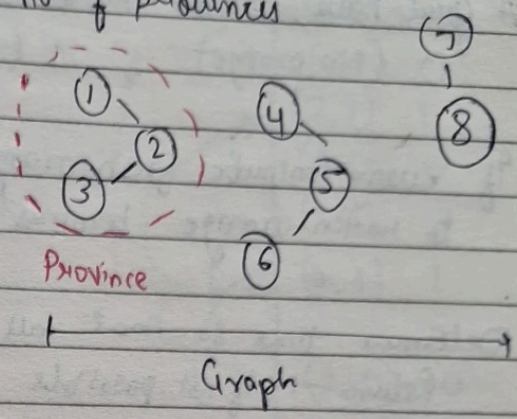
```
void DFS (int node, vector<int> adj[], vector<bool> &vis)
{
    cout << node << " ";
    vis[node] = true; //Mark true for this node

    for ( auto it : adj[node] )
    {
        if (!vis[it])
        {
            dfs(it, adj, vis);
        }
    }
```

NUBMER OF PROVINCES

Raj
DATE / /
PAGE

given matrix of connected nodes, determine no. of provinces

- given with matrix, so we will do DFS until we cover the whole matrix



- just make sure you never revisit same nodes

```
void DFS (int node, matrix, vis)
{
    vis [node] = 1 //Mark visited 1

for ( int j = 0 to matrix.Size())
{
    if (matrix [node] [j] == 1 &&
        ! vis [j] )
{
// Check it there is connection &
    not visited
{
    DFS (j , matrix , vis);
}
}
}
```

```
int findProvinces ( matrix ).
{
    int cnt = 0 ;
  Vector <int> vis (n, 0)
        initialise by NULL

for (i = 0 to matrix.size())
{       // Not visited
    if ( ! vis [i] )
    {
        DFS (i, matrix , vis); cnt++
    }
}

return cnt;
```

Rotten Oranges

A grid have number representing oranges, number are
0 (No orange), 1 (fresh orange), 2 (rotten orange)

If every minute fresh orang that is 4 directionally adjacent to rotten orange becomes rotten

t=0

2	1	1
1	1	0
0	1	1

Determine time so that all oraange becomes rotten
Return -1, if not possible

eg:

2	2	1
2	1	0
0	1	1

t=1

2	2	2
2	2	0
0	1	1

t=2

2	2	2
2	2	0
0	2	1

t=3

2	2	2
2	2	0
0	2	2

t=4

all rotten.

output: 4.

max time = 6 ∧ 2 ✓

eg:

Input:

	0	1	2
0	0	1	2
1	0	1	1
2	2	1	1

all ones are convert to twos

↓

ans = 2

Visited matrix

	0	1	2
0		2	2
1		2	2
2	2	2	2

Queue

x (r)	y (c)	time
(2,2)		2
(1,1)		2
(2,1)		1
~~(2,0)~~		~~1~~
(1,2)		1
(0,1)		1
~~(2,0)~~		~~0~~
(0,2)		0

in ↓ out

```
// Initialize all values
int n = grid.size() , m = grid[0].size();
int maxtime = 0;   ((x, y) , time)
queue < pair < pair<int,int> , int > > q;
vector <vector <int> vis ;

// Mark all rotten orange in "vis" and push them in queue
int freshcnt=0;
for ( i=0 to n)
{ for (j=0 to m)
    {
        if ( grid [i][j] == 2)
        {
            q.push ( { {i,j} , 0 } );   // level 0
            vis[i][j] = 2.
        }
    else if ( grid[i][j]==1) freshcnt ++;
}  }
```

$x + dirX[0]$ = top row
$+ \;''\; [1]$ = bottom row

DATE / /
PAGE

// easy to change direct^n via them (Top, bottom, Left, right)

```
int dirX[] = {-1, 1, 0, 0}
int dirY[] = {0, 0, -1, 1}

while (!q.empty())
{
    int row = q.front().first.first;
    int col = q.front().first.second
    int time = q.front().second;
    max time = max(maxtime, time);
    // Iterate in all 4 direction.
    for (i=0 to i=3)          // dir change by X/Y arrays
    {
        int nextrow = row + dirX[i];
        int nextcol = col + dirY[i];
        // check if all values are in bound and orange is fresh + Not vis
        if (nextrow >=0 && nextrow <=n && nextcol>=0 && nextcol<=m
            && grid[nextrow][nextcol] == 1 && vis[nextrow][nextcol] == 0)
        {
            q.push({{nextrow, nextcol}, time+1}); Push in Queue
            vis[nextrow][nextcol] = 2;   Mark Rotted
            freshcnt --;   dec the fresh count.
        }
    }
}

if (freshcnt != 0) return -1; not all oranges can be rotten
return maxtime;
```

Current elements values

Nearest 1 in Binary Matrix

Raj
DATE / /
PAGE

- Return dist of nearest 1 for each element
  - Don't use diagonal distance

Input

0	0	0
0	1	0
1	0	1

Output

2	1	2
1	1	1
1	1	1

- Initialise starting points (i.e 1) in "ans" and "vis" matrix
- Push them in Queue (coordinates and distance of nearest 1)

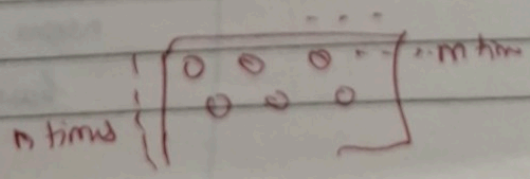
queue < x, y, steps >;

Same as previous Question (change inner for loop)

```
ans [row][col] = steps;
for (i=0 to i=3)
{
  int nextrow = row + dirX [i] ; next col = col + dirY [i]
  if (next row & next col valid & not visited)
  {
    q.push (next row, next col, step+1
    vis [nextrow] [next col] = 1.
  }
}
```

WAY TO Initialise Vectors

Vector <vector<int>> vis (n, vector<int>(m, 0));



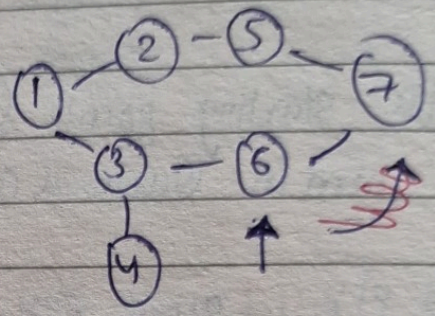
CYCLE DETECTION by BFS

DATE / /
PAGE

- Do BFS traversal and store $\boxed{\text{node, parent}}$ Queue
- Whenever a node is visited - check every times it parent is matching to the node's parent that you are traversing

Suppose vis

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



We are traversing (7, 6)
But it's already visited by (7, 5)

Now parents not match.

We are at 6 → 3, 7
3 is visited but 6 → parent = 3
So that is how 3 was visited
but for 7

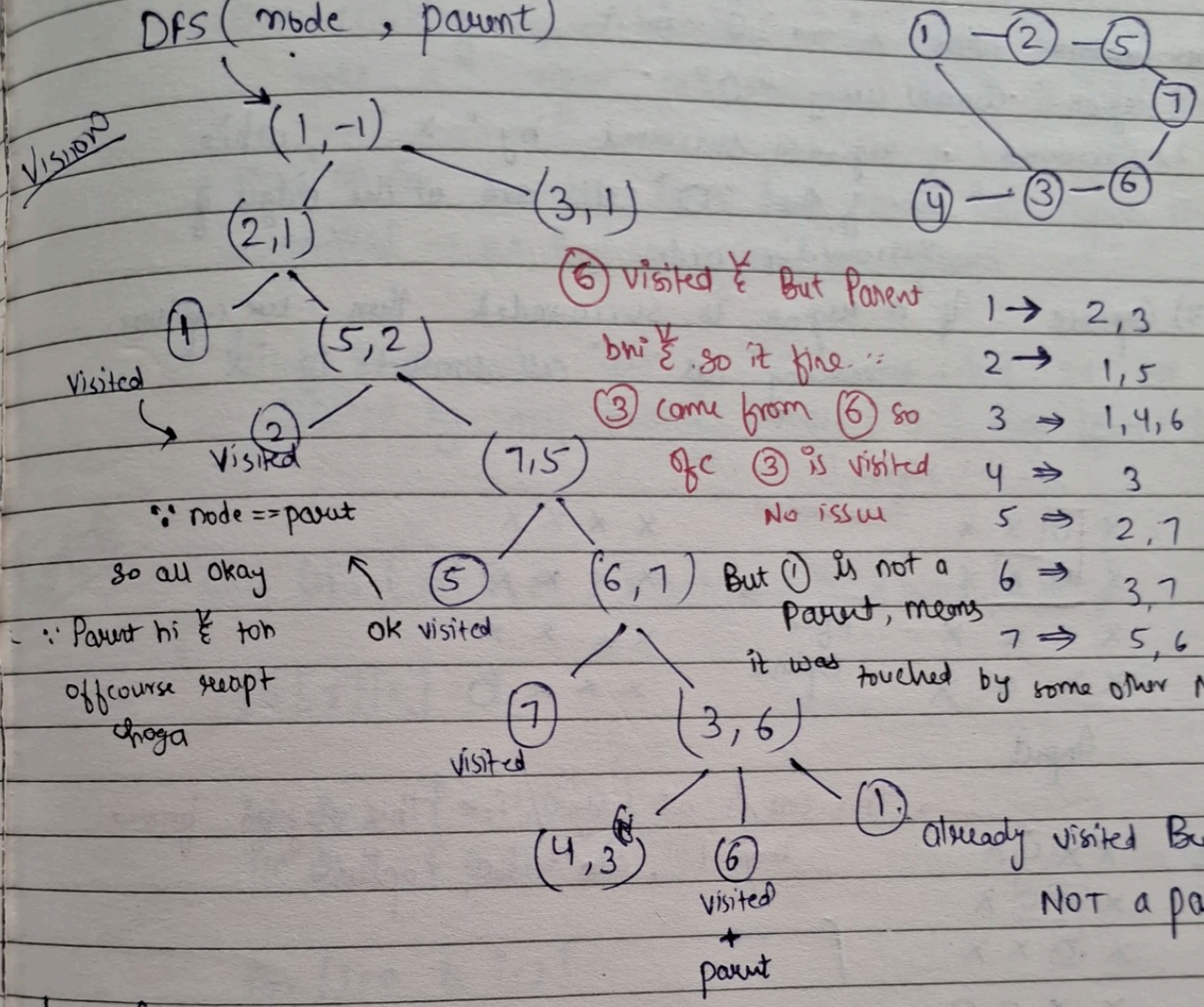
```
bool detect (src, adj[], vis[])
{
  vis[src] = 1;
  queue <pair<int, int>> q;
    q.push({src, -1});

  while (!q.empty())
  {
    int node = q.front().first;
    int parent = q.front().second;
      q.pop();

    for (auto it : adj[node])
    {
      if (!vis[it]) { vis[it] = 1
                      q.push(it, node) }
      else if (parent != it)
      {

        return true; // cycle
      }
```

how the fuck 7 was visited
means some other node have touched it

CYCLE = Visited + non Parent

CYCLE DETECTION by DFS

DFS(node , parent)



ANSWER

- Just add (node, parent) parameter in DFS traversal and if node is visited + node != parent then CYCLE is there

Capture

SURROUNDE REGION

DFS edition

- Given matrix with "X" and "O" in it
  1) Region : Connect every "O".
  2) Surround : if region in surround by "X", while none of the "O" cells are at the edges of surrounding order
  3) Capture : If a region is surrounded then we capture i.e making all "O" cell convert to "X"

this region is surrounded

eg1 :

X	O	X	X
X	O	O	X
X	X	X	X
X	X	O	X

Input

$\xrightarrow{\text{After Capture}}$

X	X	X	X
X	X	X	X
X	X	X	X
X	X	X	O

this cell is at edge. i.e this region formed by it, can't be capture

eg. 2:

X	X	O	X
X	O	O	X
X	O	X	X
X	X	O	O

After

X	X	O	X
X	O	O	X
X	O	X	X
X	X	O	O

Raj
DATE / /
PAGE

Algo

- Start checking from boundries "O" from each "O" (boundry wale) make a region and mark them as 1 in 'vis' matrix
- Now all the boundry zeros ~~have~~ with their region are marked → i.e they can never be captured
- iterate to the matrix and if "O" comes and it's not visited by any boundry zeros ("O") they convert to "X"

Code

```
void DFS (int r, c , vector< vector> & board, & vis)
{
  vis [r][c] = 1

  int dirx = {-1, 1, 0, 0}
  int dir y = { 0, 0, -1, 1}

  for (i=0 to i=3)
  {
    int nextr = r+ dirx [i] ,  nextc = c + dir y [i];
    if ( 0≤ nextr< ROWS && 0≤ nextc< COL && board [nextr][nextc] == 'O'
        && !vis [nextrow] [nextc]
    {
      DFS (nextr, nextc , board, vis);
    }
  }
```

R = Rows
C = Columns

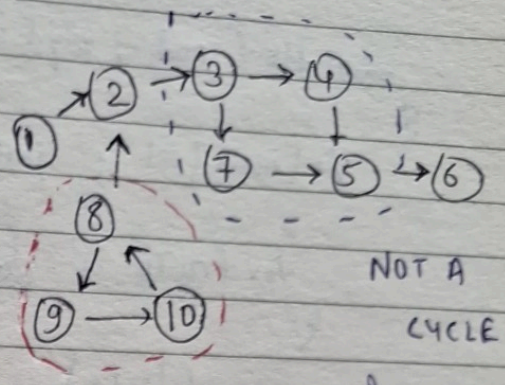
```
void Solve (& board)
{
    vector<vector<int>> vis (R, vector<int>(C, 0));

    // Search first and last Row and Make a Region
    for (i=0 to R)
    {
        if (board[0][i] == '0') DFS (0, i, board, vis)
        if (board[0][R-1] == '0') DFS (R-1, i  "  "  );

    {Same for Columns}

    // Now convert Unsafe "O" to "X".

    for (i=0 to R)
    {
        for (j=0 to C)
        {
            if (board[i][j] == "O" && !vis[i][j]
            {
                board[i][j] = "X";      // Convert to "X"
            }
        }
    }
return;
```

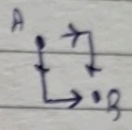
which ever "O" are not visited by boundries cell

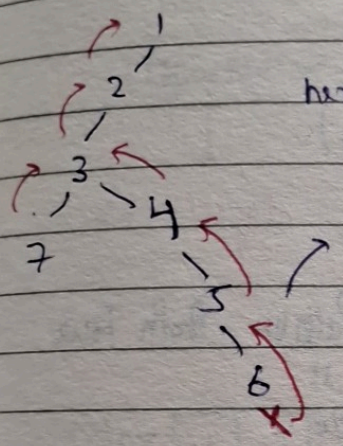
CYCLE DETECTION IN DIRECTED GRAPH

DFS

- If you have visited that node and also the path then Cycle ✓
- as you return by DFS, unmark the Path visited array

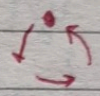




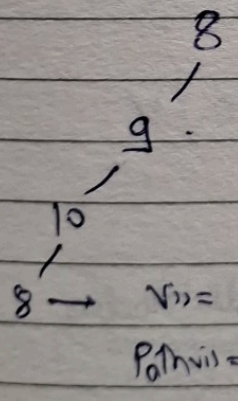


FINE

here vis = 1, Path vis = 0] so visited by diff Path



	0	1	2	3	4	5	6	7	8	9	10
vis	-1	1	1	1	1	1	1	1	1	1	1
Path vis	-1	1	1	1	1	1	1	1	1	1	1
		0	0	0	0	0	0	0			



8 → vis = 1, Path vis = 1] Cycle ✓ vis in same Path

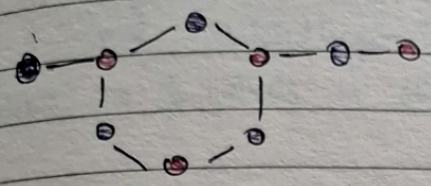
```
bool dfs ( nod , adj[] , vis[] , Pathvis[] )
{
    vis [node] = 1 ; pathvis [node] = 1

    for (auto it : adj [node])
    {
        if (!vis)
        {
            if (dfs( it, adj, vis , pathvis)
            {
            } return true
        }
        else if (pathvis [it]) // Vis and Pathvis Both True
            {                  //          ||
                return true;   //        cycle ✓
            }
    }
    Pathvis [node] = 0
    return false
}

bool cyclic (v, adj[])
                    → Iterate for all nodes
    for ( i = 0 to v )
    {   if ( !vis [i])
            if (dfs ( i ) == true)   return true
    }
}
```

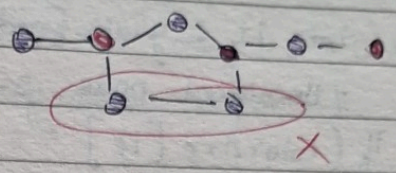
GRAPH BIPARTITE

Raj
DATE / /
PAGE

A Graph that can be colored by 2 colors but no two adjacent color should be same

eg:





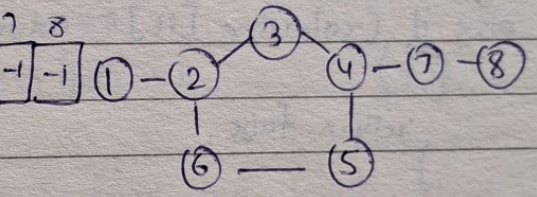
Bipartite ✓

even length Cycle

Bipartite ✗

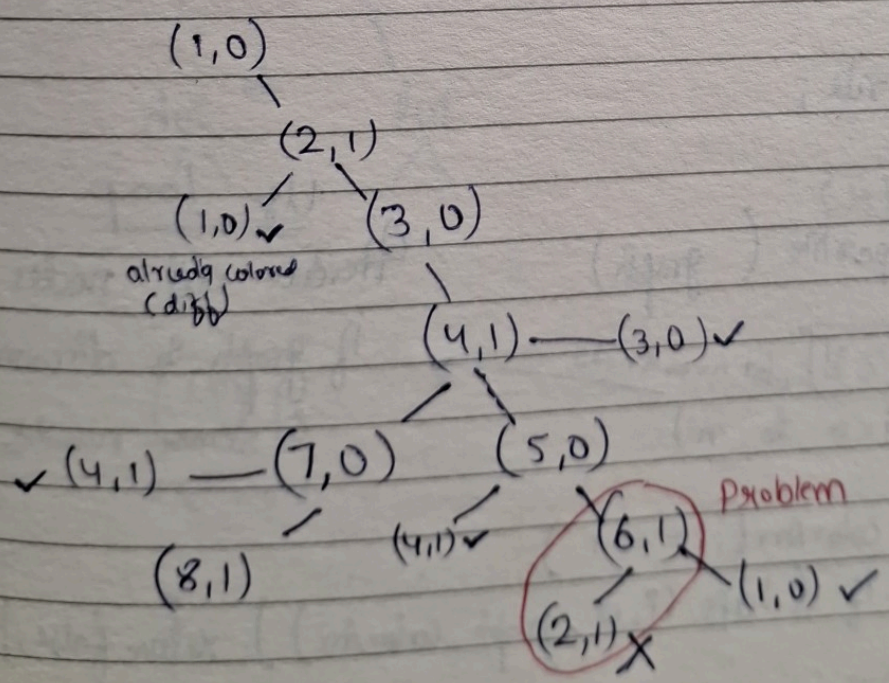
Odd length Cycle

	1	2	3	4	5	6	7	8
Color	-1	-1	-1	-1	-1	-1	-1	-1



Color < 0, 1

DFS (node, color)



1 ⇒ 1
2 ⇒ 1,3
3 ⇒ 2,4
4 ⇒ 3,5,7
5 ⇒ 4,6
6 ⇒ 2,5
7 ⇒ 4,8
8 ⇒ 7

```
bool dfs (int node, color, matrix : graph, ColorArr)
{
    colorArr [node] = color ; // color the given node (0 or 1)

    for ( int it : graph [node])
    {   // Uncolored node
        if ( colorArr [it] == -1
        {  // color with opposite color
            if (!dfs (it, 1-color, graph, colorArr))
            ① { if this call fails
                return false;
            }
        }
    Alredy colored with same color ② If already have same color
        else if ( colorArr [it] == color )
        {
            return false;
        }
    }

    return true;
}
```

Use loop.

Iterate all nodes just if graph is disconnected to some nodes

```
bool isBipartite ( graph)
{
ColorArr [-1 -1 ... -1] , int n = no of nodes
    for ( i = 0 to n)
    {
        if ( colorArr[i] == -1 )
    }   { if (!dfs (i, 0, graph, colorArr)) return false; }}
return true;
```

WORD LADDER I

Raj
DATE / /
PAGE HARD++

from word "wordbegin" to "wordend" use a dictionary "list" transform such that

- Every adj pair of word differs by a letter
- and "wordbegin" and "wordend" may not be in "list"

Return no. of words in shortest transformation sequence

eg: Start word = "hit" endword = "cog"
List = [hot, dot, dog, lot, log].

ait, bit .... zit (No exist in list) ← hit ——— (1)

hat hbt .... hzt

hot (exist in list) (take it) (2)

Try to replace all char of each place and which every word matches is (?) take that

dot lot (3)

(4)

cog (5) thus.

end word

```
int wordladder (string startword, endword, vector<str> wordlist)
{                    word, length
  queue < pair < string, int >> q;        q.push ({startword, 1});
  unordered-set <int> st (wordlist.begin(), wordlist.end());
  ↳ Set contains the word list for ease of find
  Remove those from set who was already visited
    st.erase (startword);

    while( !q.empty)
    {
      string word = q.front().first ;  int steps = q.fromt().second;
      q.pop()
      // Iterate through each letter
      for (i=0 to word.size())
      {
        char original = word[i];
          // Replace each letter from (A-Z) and check set for them
          for (char ch='a' ; ch<='z' ; ch++)
          {
            word[i]=ch
            // If word found in set
            if (st.find (word) != st.end())
            {
              st.erase;   // erase from set
              q.push(word, step+1);  // push in Queue
            }  }
          // change back to original letter
        word[i]=original
      }
```

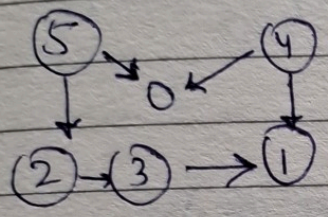
TOPOLOGICAL SORT

Raj
DATE / /
PAGE By. ~ DFS

Linear ordering of vertices such that if there is an edge between 'u' and 'v' 'u' always appears before 'v' in that ordering

* Applicable in DAG [Direct Graph + Acyclic (No cycles) graph]

eg:



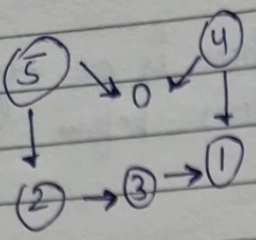
u	v
5	0
4	0
5	2
2	3
3	1
4	1

Possible linear orderings :-

[[5 4 2 3 1 0],
[4 5 2 3 1 0]
[etc]]

Vision

eg:



vis

0	1	2	3	4	5

0 → { }
1 → { }
2 = 3
3 = 1
4 = 0,1
5 = 0,2

DFS(0) DFS(1) DFS(2)

DFS(3)

5
4
2
3
1
0

Stack

DFS(4)

0 visited 1 visited

DFS(5)

0 2

- Make DFS call to all element of adjList
- Traverse by DFS and before you return from any DFS call,
  add that element to stack
- Pop element from stack = Linear Order ✓

ans = 5 4 2 3 1 0

Raj
DATE / /
PAGE

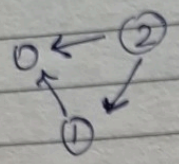
CODE

```
void DFS (int node, int vis[], stack <int> &st, adj[])
{
    vis [node] = 1
    for (auto it : adj[node])
    {
        if (!vis[it]
        {
            DFS (it, vis, st, adj);
        }

    st.push(node); // Push before you return,
    }
}

TopoSort (int N, vector <int> adj[])
{
    int vis [N] = {0};
    stack <int> st;
    Iterate for all Nodes

    for (i=0 to N)
    {
        if (!vis[i])
        {
            DFS (i, vis, st, adj);
        }
    }
}
```

— Kahn's Algorithm

TOPOLOGICAL SORT by BFS

	0	1	2
Indegree	2	1	0
	1	0	
	0		



- Print all all element with Indegree Zero.
- as you traverse (BFS) reduce the indegree of those nodes and as soon as indegree == 0

0 ⇒ { }
1 ⇒ 2
2 ⇒ 0, 1

```
vector<int> ToposortBFS ( int V , vector<int> adj[] )
{
    int indegree [V] = {0};
    // Traverse list and inc the indegree as you counter that vol
    for (int i=0 to V)
    {
        for ( auto it : adj[i])
        {
            indegree [it] ++;
        }

    queue <int> q;
    vector<int> ans;
```

```
// push all the element with Indegree = 0 in Queue
for ( int i=0 to V)
{
    if ( indegree [i] == 0)  q.push (i);
}
// Now Queue is Initialised , Start BFS
while (!q.empty ())
{
  int node = q.front ();
  q.pop();
  ans.push_back (node);

  for(auto it : adj[node])
  {
    indegree [it] - -;   reduce the edge (Indegree - -)
    if (indegree [it] == 0)
    {
        q.push (it);  push element with Indegree = 0
                      in Queue
    }
  }
}

return ans;
```

COURSE SCHEDULE

Given array of (task, prereq task) and no of task need to perform, Determine if you are able to perform given no of task

eg: (1,0), (2,1), (3,2)

Order of task: 3 2 1 0 : True

eg: (1,2) (4,3) (2,4) (4,1)

Order of task: 1 → 2 → 4 3 : False.

contradiction

- We use topological Sort : as per defination it's linear ordering of edges, so if we have (u-v) So 'u' will always come before 'v'.

If graph is DAG → topo. Sort ✓

Just check DAG

(Ans). Just write a topo Sort code and store order in Array If topo Sort contain all elements, then YES we can do all the task given to us, Since all the task can be arranged in linear Ordering.

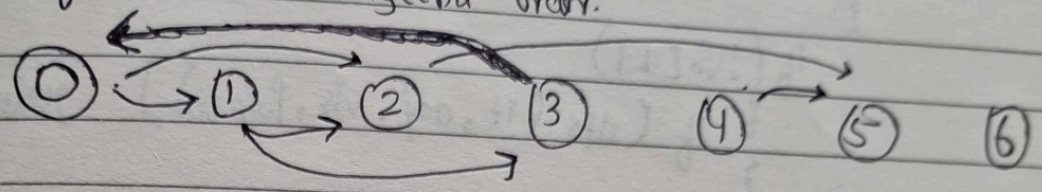
FIND EVENTUAL SAFE STATES

DFS

- Terminal Nodes : No outgoing edges from Node
- Safe Node : If every possible path starting from that node leads to a terminal node.

Return all safe nodes in ascend^n order.

eg: Input



Output: 2, 4, 5, 6

1. Any node which is part of cycle
2. Any Node leads to cycle

] CANNOT BE SAFE NOT

(Since one of thier branch will lead to themselves not termenal node)

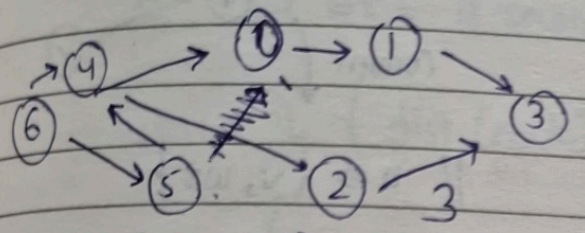
Except these 2, all are Safe nodes

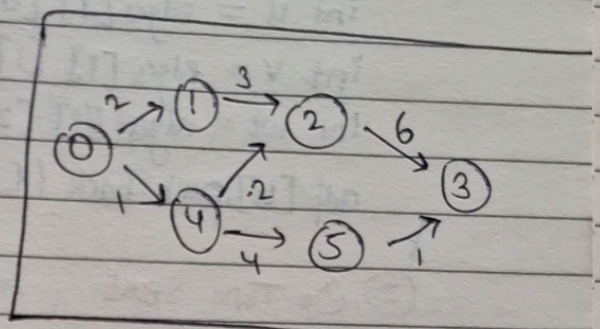
Use algorithm of Cycle detection in Directed Graph

```
bool dfs (node, adj[], vis[], pathvis[])
{
    vis [node] = 1;
    Pathvis [node] = 1;
        for ( auto it : adj [node])
        {
            if (!vis [it])
              { if ( dfs (it, adj, vis, Pathvis)) return true;
              }
            else if ( PathVis [it])     // Vis, Pathvis Both True = CYCLE ✓
              {
                return true;
              }
        } ans.push-back (node) → No cycle found
      Path vis [node] = 0     // remove Path vis as we return
        return false;

}
vector<int> SafeNode ( int V, vector adj[])
{
    int Vis [v] = {0} ,  Pathvis [v] = {0}
    Vector <int> safeNodes;
    for ( i=0 to V)
    {
        if (!vis [i])     dfs ( i, adj, vis, pathvis);
    }

}
```

SHORTEST PATH IN DIRECTED WEIGHTED GRAPH → DAG

Given: Graph with weighted edges, find shortest dist from one node to all other nodes, if not possible then -1.





{NODE DIST}

0 ⇒ {1,2}
1 ⇒ {3,1}
2 ⇒ {3,3}
3 ⇒
4 ⇒ {0,3} {2,1}
5 ⇒ {4,1}
6 ⇒ {4,2} {5,3}

$\boxed{\text{Source} = 6}$

① • Do a topo sort of graph (get topo stack)

② Take out nodes, get their adjacent, add dist of edge + parent and update the 'dist' array

Topo Stack
~~6~~
~~5~~
~~4~~
~~2~~
~~0~~
~~1~~
3

(6 d=0) → (5, d=3) → (4, d=2)

(6 d=0): +2 → (4, d=2); +3 → (5, d=3)

(5, d=3): +2 → (4, d=3+2=5) (not minimum)

(4, d=2): +1 → (2, d=3); +3 → (0, d=5)

(3 d=5): NULL

(1, d=7): +1 → 3, d=8 ×

(0, d=5): +2 → 1, d=7

(2, d=3): +3 → 3, d=6

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

Ans

finally,

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

Shortest dist arr from 6

no. of Nodes Source Edges edge (u,v,wt)

Raj
DATE / /
PAGE

```
Vector (int) ShortestPath DAG (int N, int M, vector edges[])
{      Create a Graph with weight of edges
   vector< pair <int, int >> adj [N];      < node, edge length >
   for ( i=0 to N)  ① Make Adj list
   {
      int u = edges [i] [0]
      int v = edges [i] [1]
      int wt = edges [i] [2]
      adj [u].push-back ({ v, wt })      // u => { v, wt }
   }
```

(u) —wt→ (v)

Convert

② Do Topo Sort

```
int vis [N] = {0} ;      stack <int> st.
   for (i=0 to N)
   {
      if ( ! vis [i] )   ~~topoSort~~ DFS (i, adj, vis, st);   // function Call
   }
```

③ Do Dist Calculation

```
vector <int> dist ( INT_MAX, n);      // [∞ | ∞ | ∞ | ∞] dist
dist [0] = 0   // considering source node = 0
while ( ! st.empty)
{
   int node = st.top();
   st.pop();
   if ( dist [node] ! = INT_MAX)
   {  for (auto it : adj [Node]).
```

P.T.O ⟹

}

```
if (dist[node] != INT_MAX) // Other wise just pop
{ for (auto it : adj[node])
  {
    int v = it.first;
    int wt = it.second;

    if ( dist[node] + wt < dist[v] )
    {
      dist[v] = dist[node] + wt ;
    }
  }}
}

return dist ;
}

void dfs ( node , vector(pair) adj[] , vis[] , st) // Topo Sort (By DFS) function
{
  vis[node] = 1;
  for( auto it : adj[node])
  { int v = it.first.
    if (!vis[v]) dfs( );
  }
  st.push(node);
}
```

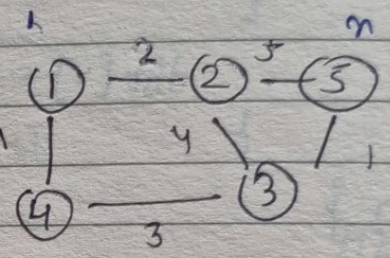
Parent dist + edge length — Previous dist

Shortest path in undirected Weighted Graph

Print Shortest Path by DIJKASTRA's ALGO

Given Undirected graph, n vertices and m edges (weighted)
find shortest path from 1 to n vertex
Return -1 if path don't exist

Input: n = 5, m = 6
edges: [{1,2,2}, {2,5,5}, {2,3,4}, {1,4,1}, {4,3,3}, {3,5,1}]
Output: 1, 4, 3, 5



- Remember where you are coming from

5 —Parent→ 3
3 —Parent→ 4
4 —Parent→ 1
Update
Path ✓

	1	2	3	4	5
Parent	1	2	3	4	5
dist	∞	∞	∞	∞	∞
	0	2	4	1	7

1 • d=0
+1 | +2
d=4 push
2 d=2 push

4, d=1
+3 | +1
3 d=4
1 d=2 ✗

(5,5)
(1,5)
~~(4,3)~~
~~(1,4)~~
~~(2,2)~~
~~{0,1}~~

Min Heap (dist, node)

2, d=2
+2 | +4 | +5
3 d=6 ✗ | 5 d=7

3, d=4
+4 | +3 | +1
2 d=6 ✗ | 4 d=7 ✗ | 5 d=5 ✓

5 d=5

5, d=7

(u, v, wt)

```
vector<int> SH (int m, n, matrix edges)
        Make adj list of size n+1 (since 1 based indexing)
{ vector <pair <int, int>> adj [n+1];
    for (auto it : edges)
    {
        adj [it[0]].push-back ({ it[1], it[2]})
        adj [it[1]] .push-back ({ it[0], it[2]})
    }

    priority-queue ——— min Heap ——— pq.
    vector <int> dist (n+1, INT-MAX);
    vector <int> Parent (n+1);
    for (i=0 to n+1) parent [i] = i;  → // declare values = index itself
    dist [1] = 0      Initialize source

    while ( ! pq.empty ()) {
        auto it = pq.top();
        int node = it.second        int dist = it.first
        pq.pop();

        for (auto it : adj [node]
        {
            int adjnode = it.first
            int edgewt = it.second
            if (dis + edgewt < dist [adjNode])
            {
                dist [adjnode] = dis + edgewt
                pq.push ({dist [adjnode], adjNode})
                parent [adjnode] = node
            }
        }
    }

    if (dist[n] == INT-MAX)   → If path not found
        return {-1};

    Make Path from Back
    vector <int> path; int x = n;
    while (parent [x] != x)
    {
        Path.push (x)
        x = parent [x];
    }
    path.push (1);
    return reversed Path
```

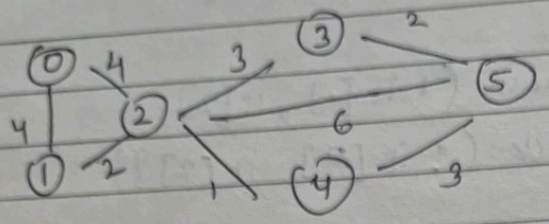
this condition added in Dij algo

Undirected Weighted Graph

DIJKTRA'S ALGORITHM.

only → Priority Queue

Source = 0



0 ⇒ {1,4} {2,4}
1 ⇒ {0,4} {2,2}
2 ⇒ {0,4} {1,2} {3,3} {4,1} {5,6}
3 ⇒ {2,3} {5,2}
4 ⇒ {2,1} {5,3}
5 ⇒ {2,6} {3,2} {4,3}

Prio Queue pop

0, d=0 → 1, d=4 (+4); 2, d=4 (+4)

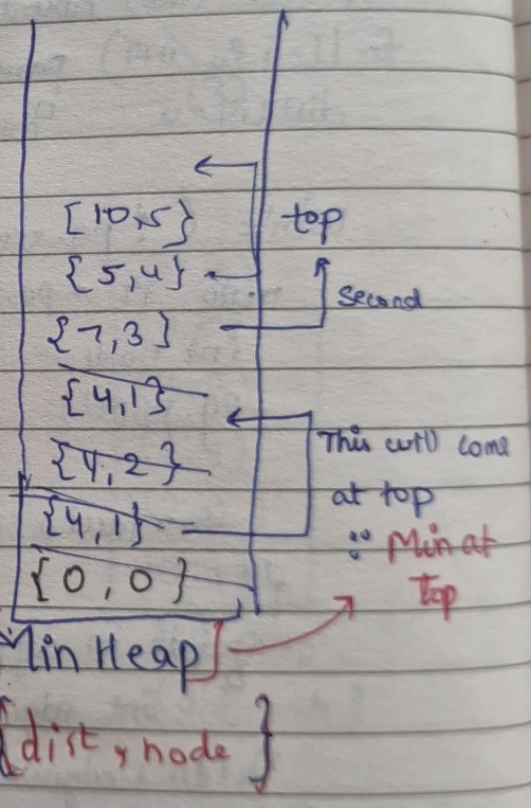
1, d=4 → 0, d=8 ✗; 2, d=6 ✗

2, d=4 → 0, d=8 ✗; 1, d=6 ✗; 3, d=7; 4, d=5; 5, d=10

4, d=5 → 2, d=6 ✗; 5, d=8 ✓

3, d=7 → 2, d=10 ✗; 5, d=10 ✗

5, d=10 → 3, d=10 ✓; 4, d=13 ✗



0	1	2	3	4	5
∞	∞	∞	∞	∞	∞
0	4	4	7	5	10

dist

answer ↓

0	1	2	3	4	5
0	4	4	7	5	8

Weighted Graph

Graph don't contain -ve wt cycle.

```
vector<int> dijkstra(int V, matrix adj[], int s)
{   // min Heap
    priority-queue <pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>> pq;

    vector<int> dist(V, INT_MAX);
    dist[s] = 0;   dist of source = 0
    pq.push({0,s});

    while(!pq.empty())
    {
        int dis = pq.top().first;
        int node = pq.top().second;
        pq.pop()
                    // traverse that node
        for(auto it: adj[node])
        {
            int edgewt = ~~it.first~~ it[1]
            int adjNode = it[0];

            if(dis + edgewt < dist[adjNode])
            {
                dist[adjNode] = dis + edgewt
                pq.push({dist[adjNode], adjNode});
            }
        }
    }
    return dist.
}
```

Parent edge length prev value

SH in Binary Matrix

find Shortest path between source and destination cell, path can only be created if value is 1

Src = (0,1) desti = (2,2)

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

we don't need priority Queue since there is not random wt edges, we will alway increment by 1 unit so, Queue will work ✓.

dist matrix

Src

∞ 1	0 ∞	∞ 1	∞ 2
∞ 2	∞ 1	∞ 2	∞ 3
∞	∞	∞	∞
∞	∞	∞	∞
∞	∞	∞	∞

0, (0,1)

Queue.

(dist, {row, col})

finally you will come at (2,2) destination and dist on cell will be Shortest distance

```
int function ( matrix: grid , source , destination की pair (x,y))
{
    queue <pair <pair <int, int>> q;    int n = grid.size() , m = grid[0].size();
    vector <vector <int>> dist ( n, vector<int>(m, INT-MAX) );

    q.push ({0, {source.first , source.second } });
    int dirX[] = { -1 , 1 , 0 , 0 } ;
    int dirY[] = { 0 , 0 , -1 , 1 } ;

    while ( ! q.empty() )
    {
        auto it = q.front() ;   int dis = it.first , r = it.second.first , c = it.second.second
        for (i=0 to 4)
        {
            int newr = r + dirX [i] , newc = c + dirY[i] ;
            if ( newr and newc in Bounds)
            {
                if ( grid [newr] [newc] == 1 && dis+1 < dist [newr] [newc])
                {
                    dist [newr] [newc] = dis + 1 ;
                    // check if you are at destination or not
                    if ( newr & newc == destination x, y)
                    {
                        return (dis+1);
                    }
                }
            }
        }
    }
    We never encountered destination coordinates
    return -1
```

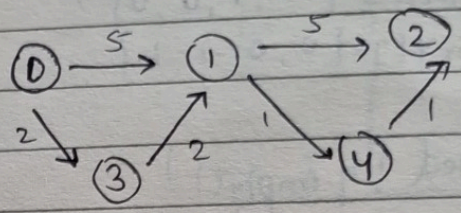
Cheapest flight within 'k' steps

"n" cities connected by some flights [] = {from, to, price}

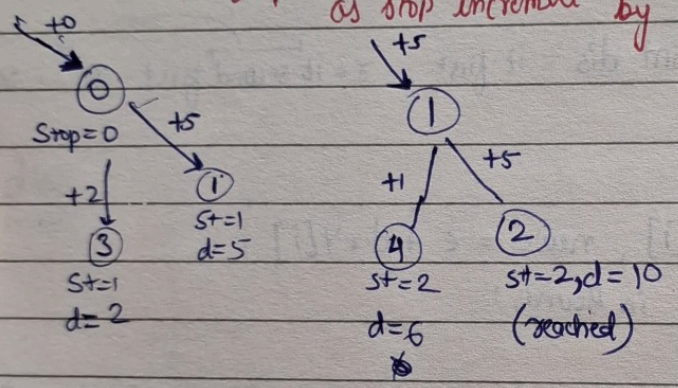
Return cheapest flight from 'src' to 'dest' within 'k' stops
(stops are node excluding src and dest.)

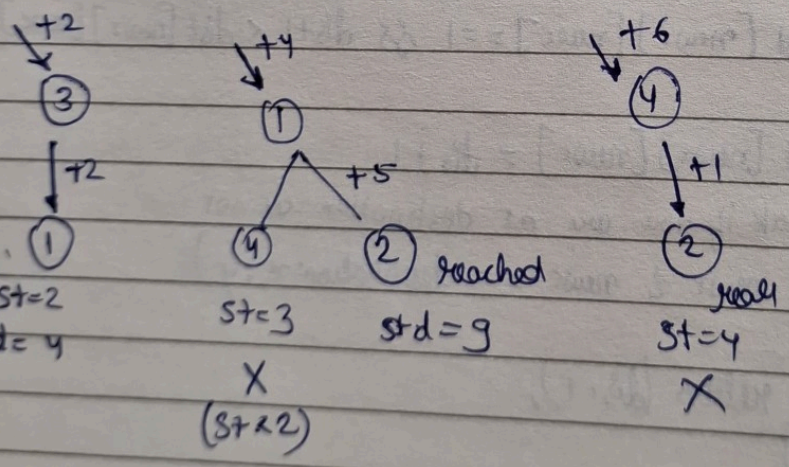
Here no. of stops is our priority rather than distance.

No need of priority Queue as stop increment by 1 unit –



Src = 0
dest = 2
K = 2





(3, 2, 7)
(2, 4, 6)
(2, 1, 4)
(1, 3, 2)
(1, 1, 5)
(0, 0, 0)

(Stops, node, dist)

	0	1	2	3	4
dist.	∞	∞	∞	∞	∞
		~~5~~ 4	~~10~~ 6	2	6

dry run Yourself

{u, v, wt}

↑

```
vector funct (int n, matrix flights, int src, dest, int k)
    Make Graph
{
    vector<pair> adj[n];   // node, wt
    for (auto it : flights)
    { adj[it[0]].push_back({it[1], it[2]})
    }
    queue.pair<pair> q
      q.push({0, {src, 0}});
      vector dist (n, INT_MAX);
      while (!q.empty())
      {
          auto it = q.front; q.pop()
          stops = it.first , node = it.second.first, cost = it.second.second
          if (stops > k) continue
          for (auto it : adj[node])
          {
              int adjnode = it.first
              int wt = it.second
              if (cost + wt < dist[adjnode] && stops <= k)
              {
                  dist[adjNode = cost + wt;
              }

          }
      }
      if (dist.[dest] == INT_MAX) return -1

      return dist[dest];
```

Raj

DATE / /

PAGE

Basic Point to Remember. (In Qs of Shortest Path)

1) Choose b/t Queue & Priority Queue.

↳ Use priority Queue if you ~~feel~~ have random edge weight. and whichever parameter you want min store it in first.

2) Make List/array of size (n+1) when there is 1 Based Indexing.

No. of ways to arrive in min time

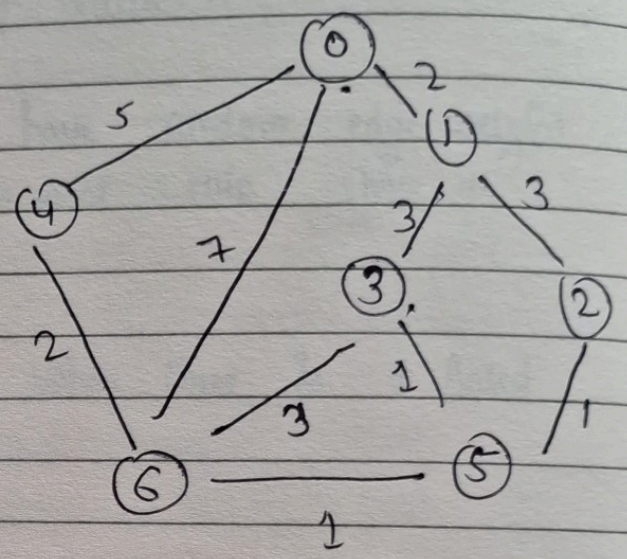
Count no of ways you can travel to (n-1) from 0 in min time

Input $n=7$, SRC=0, dest=6

Output: 4

Explain
- $0 \to 6$
- $0 \to 4 \to 6$
- $0 \to 1, 2 \to 5, 6$
- $0 \to 1 \to 3 \to 5 \to 6$

all reach (6) in min time = 7



ALGO

KEY: Always add the ways of parent + (1) current for each node, that's how many ways you can reach a particular node

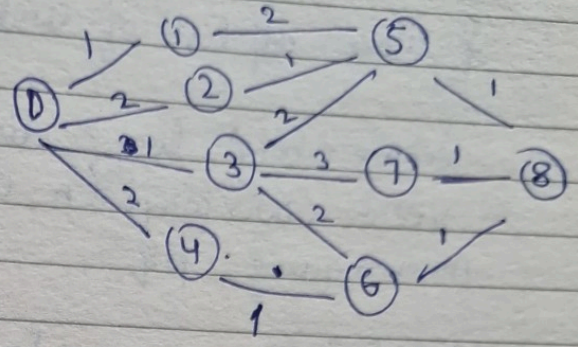
Raj
DATE / /
PAGE

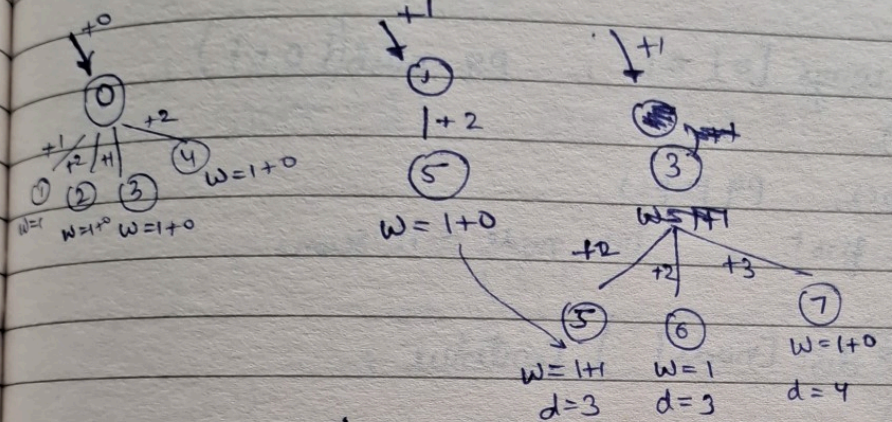
Src = 0 , dest = 8

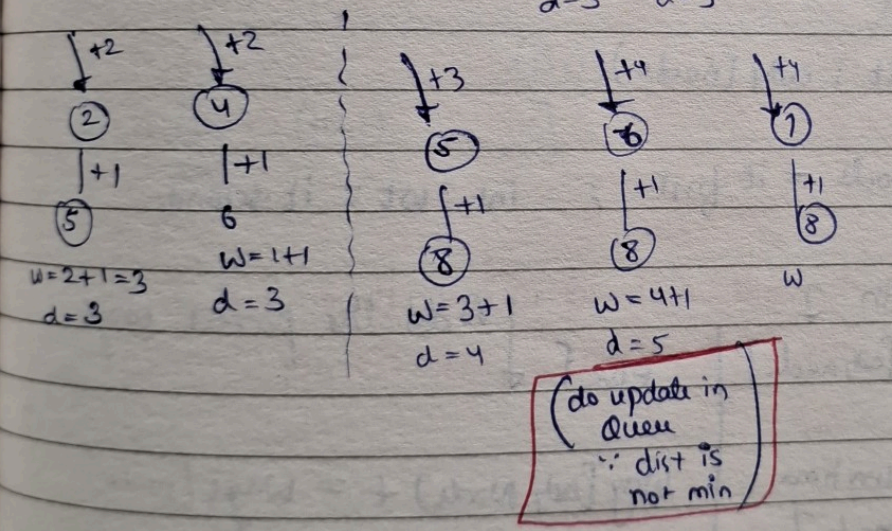
VISION

	0	1	2	3	4	5	6	7	8
Ways	1	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8
dist	0	∞	∞	∞	∞	∞	∞	∞	∞







(do update in Queue ∵ dist is not min)

Pop Sequence due to min Heap

(4,8)
~~(3,6)~~
(3,6)
(4,7)
(3,5)
~~(2,4)~~ 4th
~~(1,3)~~ 2nd
~~(2,2)~~ 3rd
~~(1,1)~~ 1st
~~(0,0)~~

Min Heap
{ dist , node }

	0	1	2	3	4	5	6	7	8
Ways	1	0	0	0	0	0	0	0	0
		1	1	1	1	1	1	1	4
						2	2		
						3			

	0	1	2	3	4	5	6	7	8
	0	∞	∞	∞	∞	∞	∞	∞	∞
		1	2	1	2	3	3	4	

Raj
DATE / /
PAGE

```
vector funct^n (int n, matrix : roads)
{ const int MOD = 1e9 + 7

  { Make adj list of undirected graph : adj }

  vector <long, long> dist (n, LLONG_MAX);
  vector <int> ways (n, 0);
  priority queue pq

  dist [0] = 0 ; ways [0] = 1 ; pq.push({0,0}) ;
  while (!pq.empty())
  {  auto it = pq.top(); pq.pop()
     int currtime = it.first ; int node = it.second

     if (currtime > dist [node] ) continue ;

     for (auto it : adj [node])
     {
        int adjnode = it.first ; int wt = it.second;

// Found a new min Path
if ( wt + currtime < dist [adjnode] )
{
   dist [adj node] = wt + currtime
   way [adj node] = way [node]
   pq.push ({ )
}

else {
   Way [adj Node] += Ways [node]
   No need to push as it is not
   a min path

Add the parent ways.

return ways [n-1]
```

MINI MULITIPLICATIONS

Raj
DATE / /
PAGE

Start = 3 end 30

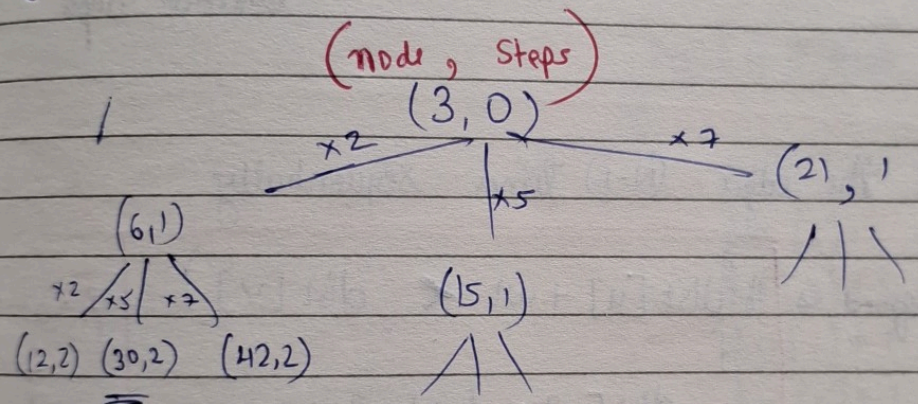
Now min no of multiplications so that you reach end (multiply nodes)

mod by 100000 if no exceeds

Input { arr[] = {2,5,7}
start = 3 end = 30

output : 2

Eg : Start = 3 arr = [2,5,7] end = 30



Application of Dijkstra's Algo (without priority Queue)

But node value be less than 99999 if it exceeds 100000 then mod by 100000

APP

0	1	2	3	4	. . .	6	. . .	99999
∞	∞	∞	∞	∞	.	6	.	∞ ∞
			0			1		

(1,21)
(1,15)
(1,6)
(0,3)

Steps, node

ans is whenever you reach that no = Step+1

BELLMAN FORD: Shortest Path Algorithm

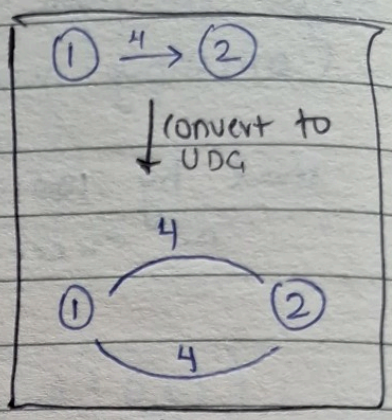
NEED

Djkstra's can't be implemented for -ve edges (DRAWBACK)

Bellman Ford helps to detect -ve cycles

Only for UnDirected Graph (DG)

BF Algo is src to destination
(Single Source Shortest path Algo)



How to Implement in Directed Graph

STATEMENTS

→ Relax all the edges (N-1) times sequentially

RELAX EDGES defined as

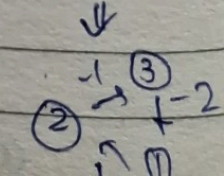
```
if(dist[u] + wt < dist[v])
{
    dist[v] = dist[u] + wt
}
```

Relaxation of edges

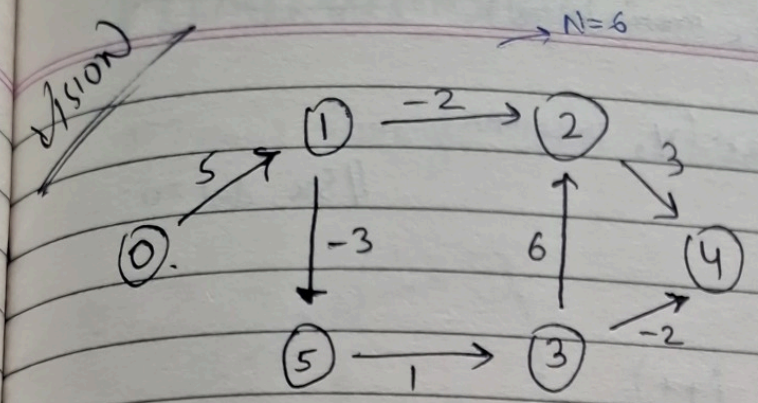
- WHY (N-1) times
  ↳ Since edges given can be in any Order, cuz considering Worst case we might get 1st edge value at last

- HOW TO DETECT -ve CYCLES
  Since we only need (N-1) iteration for relaxation but in Nth Iteration if cycle anything get relaxed then ⇒ -ve cycle

-ve Cycle eg:
Sum of all edge wt is -ve



Vision



can be in any Order

(u v wt)

Sequence

(3, 2, 6)
(5, 3, 1)
(0, 1, 5)
(1, 5, -3)
(1, 2, -3)
(3, 4, -2)
(2, 4, 3)

We do relaxation (N-1) times and update dist[] (Sequentially)

	0	1	2	3	4	5
dist	0	∞	∞	∞	∞	∞
		5	3	3	6	2
					1	

loop (N-1) times

{

dist[3] + 6 < dist[2]

dist[5] + 1 < dist[3]

dist[0] + 1 < dist[1]

dist[1] + 1 < dist[5]

dist[1] + 1 < dist[2]

dist[3] + 1 < dist[4]

dist[2] + 1 < dist[4]

}

Relaxation

(dist[u] + wt < dist[v])

{

dist[v] = dist[u] + wt

}

* finally, the dist[] array gives Shortest distance from src to all Nodes

(u, v, wt)

Raj

DATE / /

PAGE

```
vector<int> Bellmanford (int V, S, matrix edges)
{
    // dist array.   vector<int> dist (V, INT-MAX);
                                          // Src dist = 0
    dist[0] = 0;
    // Do (N-1) relaxation

    for (i=0, i<V-1 ; i++)
    {
        for (auto it : edges)
        {
            int u = it[0]
            int v = it[1]
            int wt = it[2]

            if (dist[u] != INTMAX && dist[u] + wt < dist[v])
            {
                dist[v] = dist[u] + wt;
            }
        }
    }

    // Do Nth traversal to Detect -ve cycle   (if yes : return -1)

    for (auto it : edges)
    {
        if (dist[it[0]] != INTMAX && dist[it[0]] + it[2] < dist(it[1]])
        {
            return -1;
        }
    }

    return dist;
```

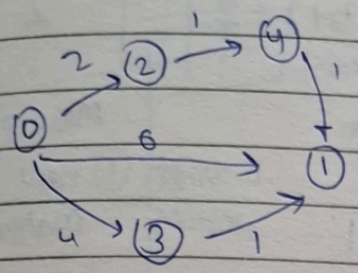
FLOYD WARSHALL : Multiple source Shortest Path Algo

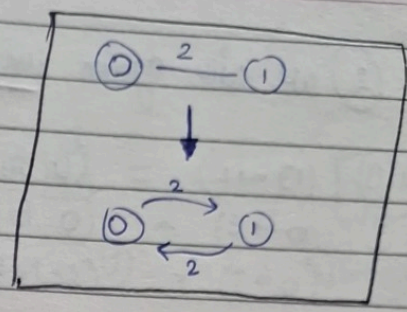
+ (Detect -ve cycles)

Raj
DATE / /
PAGE

Travel through each vertex

eg:





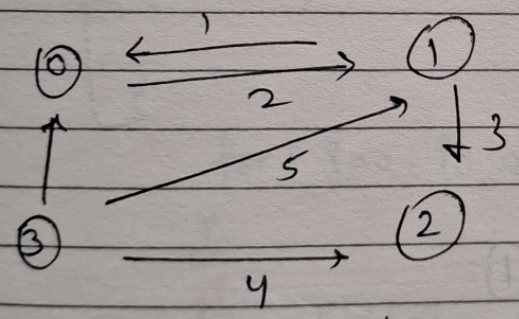
convert Undirected to directed for this Algo

$$G_0\ (0 \text{ to } 1) = \begin{cases} (0 \text{ to } 2) + (2 \text{ to } 1) = 5 & (\text{via } 2) \\ (0 \text{ to } 3) + (3 \text{ to } 1) = 5 & (\text{via } 3) \\ (0 \text{ to } 4) + (4 \text{ to } 1) = 4 \checkmark & (\text{via } 4) \end{cases}$$

[Path is min ((i to k) + (j + k) = SP (i to j))]

Q.

eg.



(1) Make a dist / cost Matrix from given graph

eg: (1) $\xrightarrow{3}$ (2)

Cost of [1][2] = 3

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

DATE / /
PAGE

(2) we will again use already matrix to compute distance

via (0) · (0→1) = (0 to 0) + (0 to 1) = 0+2 = 2
via (0) (0→2) = (0 to 0) + (0 to 2) = ∞
via (0) (0→3) = (0 to 0) + (0 to 3) = ∞

via (0) (1 to 0) = (1 to 0) + (0 to 0) = 1
via (0) (2 to 0) = 0
via (0) (3 to 0) = 0

via (0) (1 to 2) = (1 to 0) + (0+2) = 1+∞ = ∞
via (0) (1 to 3) = (1 to 0) + (0 to 3) = ∞
via (0) (2 to 1) = (2 to 0) + (2 to 1) = ∞
via (0) (2 to 3) = ∞

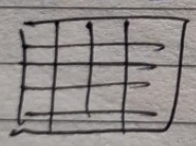
" (3 to 1) = 5
" (3 to 2) = 4
"

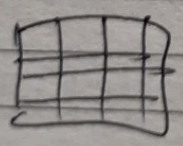
Make a new matrix

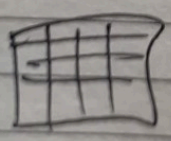
0	2	∞	∞
1	0	3	∞
∞	∞	0	∞
3	5	4	0

via (0)

Similarly →

via (1) ≡  via (1)

 via (2)

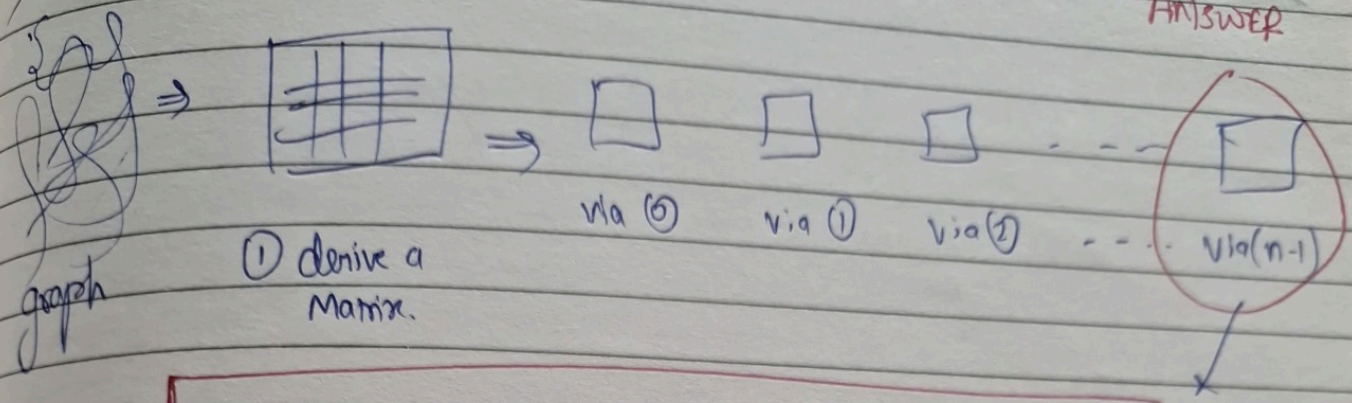
 via (3)

pi

Vision

Raj
DATE / /
PAGE

ANSWER



finally editing it, you will get matrix which have shortest distance of all nodes from each vertex as a source.

```
for ( via = 0 to N)
{
    for ( i = 0 to n)
    {
        for ( j = 0 to n)
        {
            cost[i][j] = min( cost[i][j],
                              cost[i][via] + cost[via][j])
```

#DETECT -ve cycle. $\boxed{\text{if cost[i][i] < 0}}$ instead of zero

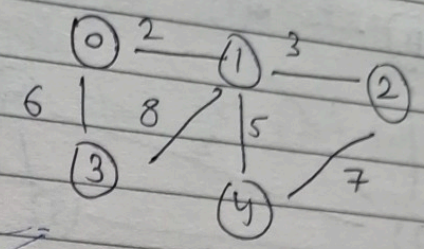
↓

return {-1}

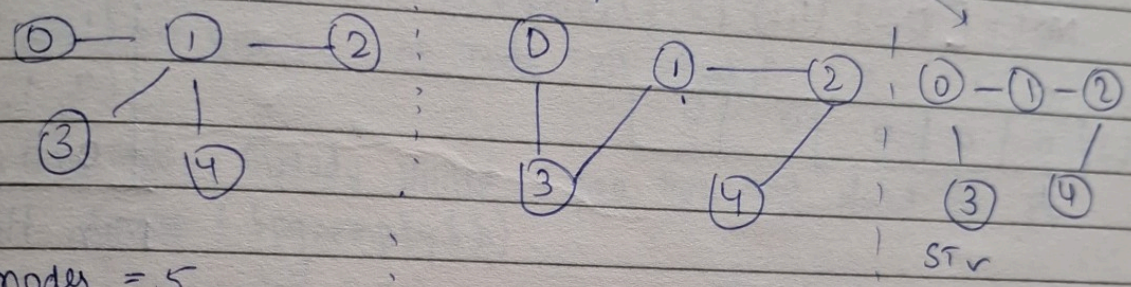
Minimum SPANNING TREE : MST

Spanning Tree ① Tree with
{ 'n' nodes
"n'-1" edges }

② All nodes should be reachable from eachother



eg:



nodes = 5
edges = 4
(Spanning Tree ✓)

ST ✓

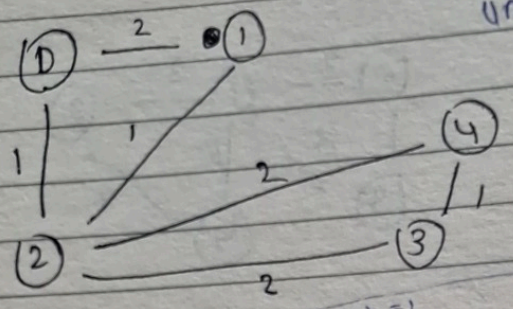
ST ✓

etc.

• Spanning Tree with least sum of edge wt = $\boxed{\text{MST}}$

PRIM's ALGO.

↳ Find min MST of connected, weighted undirected Graph



Sum = 0 → 2 → 4 → 5

wt = 1

MST = [(0,2) (1,2) (2,3) (3,4)

	0	1	2	3	4
vis[]	0	0	0	0	0

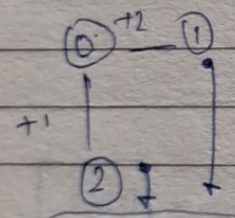
(2,2,3)
(1,4,3)
(2,3,2)
(2,4,2)
(1,1,2)
(1,2,0)
(2,1,0)
0,0,-1

→ Start Pt

Min Heap

Wt, Node, Parent

lets start with 0



don't mark 2, 1 as vis

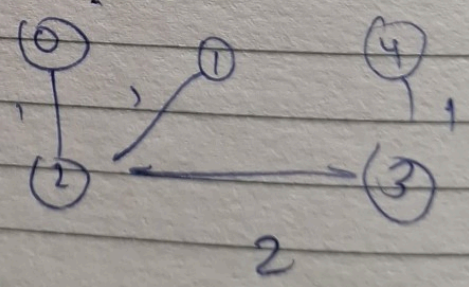
Job pop karega tab mark karna

Similar do all

we get

MST [(0,2) (1,2) (2,3) (3,4)]

↓ wt



Sum of edge = 5 ✓

Q Find sum of edges of MST

```
int func^n (int v, adj[])
{
  minheap : pq ;  vector <int> vis (v, 0);   int sum=0
  pq.push({0,0}); // push (wt, node)
No need for storing parent if not asked about MST (here only sum needed)
  while (!pq.empty())
  {
    auto xit= pq.top();   pq.pop();
    int wt =xit.first ;   int node =xit.second,

      //if node already visited = pop then next element from pq
        ~~while (!pq.empty())~~
      if (vis[node] ==1)   continue

else //. add its wt to the sum (include in MST)
      sum += wt
      vis[node] =1,

        iterate on all adj node
        for (auto it : adj[node])
        {
          int adjnode = it.first;   edgewt = it[1];
          push in Queue if not visited
            if (!vis[adjnode])
            {
              pq.push(edgewt , adjnode);
            }
        }
  }
}
return sum;
```

TC = O(logE + E logE)

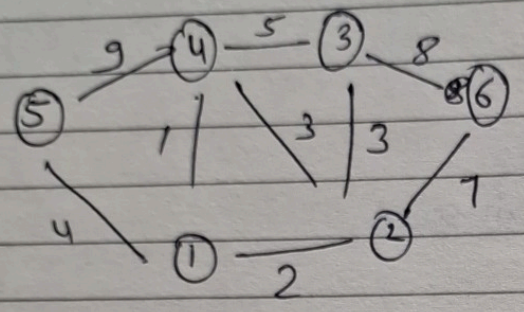
(Read disjoint set → →)

Raj

DATE / /
PAGE

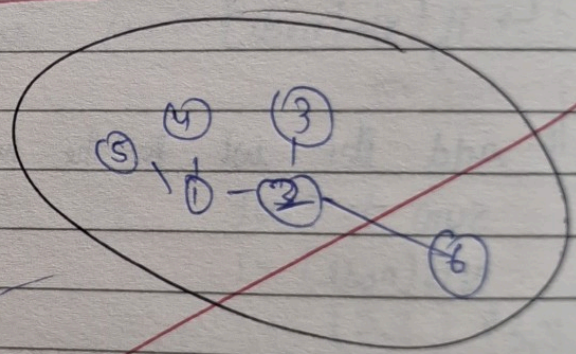
KRUSKAL'S ALGORITHM

↳ Helps to find MST



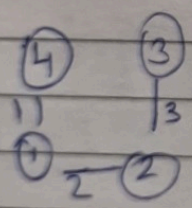
Take	wt	u	v
✓	1	1	4
✓	2	1	2
✓	3	2	3
✗	3	2	4
✓	4	1	5
✗	5	3	4
✓	7	2	6
✗	8	3	6
✗	9	4	5

① Sort all the edges acc^n to edge wt



MST

→ till now all were not part of single component



← 2, 4 since this is already part of single component leave it

BCoz, No need to include extra edge and make cycles, we just need MST

Sum = 1+2+3+4+7

CODE

```
int MST-KA (int V, adj[])
{
    vector <pair < pair >> edges (wt, u, v)
  for( i=0 to v)
  {
    for (auto it : edges[i])
    {
      int adjnode = it[0]
      int wt = it[1]
      int node = i

    edges.pushback (wt, { node, adjnode } )
No need to push 2 times as it will be ignored
    }
  }

// Sort acc^n to edge wt
  Sort (edges.begin, edges.end)
  DisjointSet ds(V) ;  ——— next pg.
// MST sum=0 at start
  int MST = 0 ;

for( auto it : edges)
{
  int wt = it[0]
  int u = it[1]
  int v = it[2]
```

Raj
DATE / /
PAGE

not part of same component

```
  if ( ds.findUltipar(u) != ds.findUltipar(v) )
  {
    So add wt to MST sum
    MST += wt
  }
}

return MST ;
```

Connecting Graph

- Kruskal's Algo

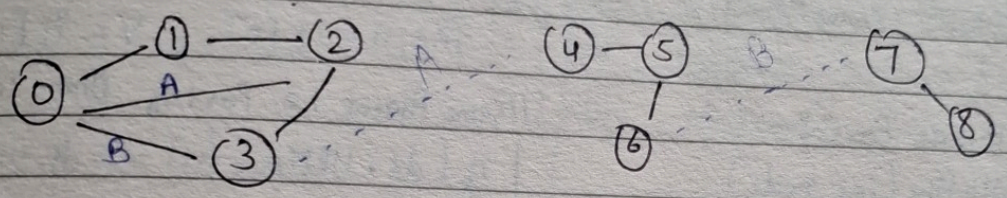
DATE / /
PAGE

given graph : 'n' vertices and 'm' edges

You can remove any edge and put b/t any 2 vertices

find min number of operation to make graph connected

eg:



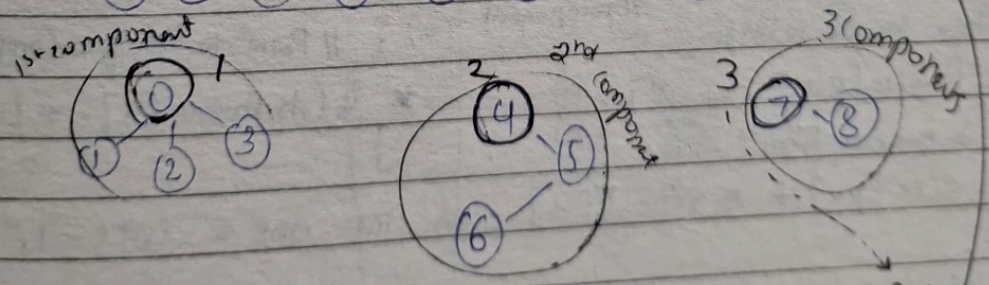
Sol^n) remove A and connect (3)-(4)] Ans = 2
remove B " " (6)-(7)

- For N connected comp. we need (n-1) edges to connect them
- Count extra edges — if (extra edges >= (n-1)) ans = n-1
  else — ans = -1

- Write all edges

(0) (1) (2) (3) (4) (5) (6) (7) (8)

(0,1) ✓
(0,3) ✓
(0,2) ✓
(1,2) ✗
(2,3) ✗
(4,5) ✓
(5,6) ✓
(7,8) ✓



these element were already in same system

= 2 = 3-1

No of ignored edges = extra edges

★ When: Vertex pointing to itself (Parent[])
↳ = component.

```
Class DisjointSet {
public:
vector<int> Parent, size, rank;
//initialize size/Parent by constructor
DisjointSet (int n)
{ rank.resize (n,0)
  size.resize (n,1)
  Parent.resize (n);
for (i=0 to n) Parent[i] = i;
}
Find Ultimate Parent
int UltiPar (int node)
{   //Point^n to itself Ultimate Parent
if (node = Parent [node]) return node
else
return Parent [node] = UltiPar (Parent [node]);
}
Union By Size
Void UnionbySize (int u, int v)
{
int pu = ds.UltiPar(u), pv = ds.UltiPar (v);
if (pu = pv) return   // Already Same // component
else Join smaller to larger
if (ds.size [pu] < ds.size [pv])
{ ds.Parent [pu] = pv
ds.size [pv] + = ds.size [pu];
}
else {
ds. Parent [pv] = pu
ds.size [pu] + = size [pv];
}
}
```

```
Class Solution

int Solve (int n, matrix edge)
{
  DisjointSet ds (n);   constructor
  int cntextra = 0      extra edges counter

  for (auto it : edge)
  {
    int u = it [0], v = it [1]
  //Same Parent i.e. Part of One component
  if (ds.Ultipar (u) == ds.Ultipar (v))
  { ignore them ∵ already in same component
    cntextra ++;   // extra edge
  }
  else { //else: join them
    ds.unionBySize (u,v);
  }
}
// Count connected Component = cntC
  int cntC = 0
  for (i = 0 to n)
  { // Point^n to itself = Component
★ if (ds.Parent [i] == i) cntC++;
  }

  int ans = cntC - 1;   ans = (connected comp - 1)

  if (cntextra >= ans) return ans;
  else return -1;
```

Account Merge

Merge the similar accounts (same name, email)
Return merged acc in sorted order : But acc themselves can be in any order

{accounts belong to same person if there is a common email}

Index
0 → [John, J1@com, J2@com, J3@com]
1 → diff [John, J4@com]
2 → [raj, r1@com, r2@com]
3 → Same [John, J1@com, J5@com]
4 → guy [raj, r2@com, r3@com]
5 → [Mary, M1@com]

same guy

finally,

→ SORTED. (must)

[John, J1@com, J2@com, J3@com, J5@com]
[John, J4@com]
[raj, r1@com, r2@com, r3@com]
[Mary, m1@com]

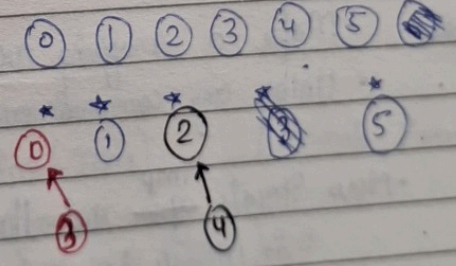
Account in any order

• Data structure that help to merge (dynamically) = Disjoint Set

Assume Indices of each Row as each node and traverse

J1@com, J2@com, J3@com = 0
J4@com = 1
r1@com = 2
r2@com = 2
J5@com = 3
r3@com = 4
m1@com = 5

connect
J1@com
r2@com



• Now merging the mails, merge element of each node to its parent element.

like all (3) element (string) merge with (0) elements

STEP (2)

0 =	J1	J2	J3	J5
1 =	J4			
2 =	r1	r2	r3	
3 =	{ }			
4 =	{ }			
5 =	m1 •			

(Com

STEP-3

Sort each row and put name ahead to get answers

```
Class DisjointSet {
    [COPY PREVIOUS CODE]
};
```

~~Class Sol{~~

STEPS

~~Vector~~ • Make a mapping of all emails to index no.

• Union any common email to it's parent

• Now iterate ~~again~~ map and Union (Path compress) of childs to their Ultimate parent

name always at start ← (name, email1, email2 ....)

Raj
DATE / /
PAGE

```
vector<vector<string>> mergeAccounts (matrix<string> details)
{ ① Map and Join ~~et~~ nodes.
  int n = details.size();
  Unordred_map <string, int> map          // email, node
  for(i=0 to n)
  { for(~~auto~~ j=1 to details[i].size)
    {
      string mail = details[i][j]
      if(map.find(mail) == map.end())  // new email
      {
        map[mail] = i;
      }
      else  merge/union/connect to its parent
      {
        ds.unionbySize(i, map[mail]);
      }
  ② Now Union/Path compression of all children node to
     their ultimate parent
  vector<string> mergedmail[n];
  for(auto it : map)
  {
    string mail = it.first
    int node = it.second
    int pnode = ds.ultiPar(node);      //ultimate Parent of node
    // Push string in ultimate node
    mergedmail[pnode].push_back(mail);
```

Raj
DATE / /
PAGE

vector<vector<strings> ans;

SORTED →

③ Sort and Rearrang in format : name , email 1 , email 2

```
for ( i=0 to n)
 {
    if(mergedmail [i] . size () ==0)  continue

    Sort ( merged mail .begin () , mergedmail . end () ); // Sort mails.
    Vector < string > temp ;
    temp . push_back (details [i][0]), // push name in a string

    //now push all the sorted emails
      for (auto it : mergedmails [i] )
      {
        temp . push_back (it) ;
      }
  ans . push_back( temp );
 }

return ans.
```