graph

lect-1 (5/10/20).

① (dfs).
- const
- display()

② Remove edge, get edge.
③ Remove vtx.
④ has path.
⑤ Print all path.
⑥ max wt / min wt.

lecture-2 (7/10/2020).

① Hamiltonian path and cycle
② No. of islands (1200)
③ max. area of island (695)
④ max perimeter of island (403)
⑤ surrounded region (130)
⑥ No. of distinct island (694). (H.W)

lect-3 (9/10/2020) [bfs]

① Bfs traversal
- ① Normal bfs (just put and remove). level)
- ② Bfs with null pointer (put null at each)
- ③ Bfs with cycle (dist put in arr).
- ④ Bfs without cycle (same as ③ just mark before call).

② 785 (is graph bipartite).
③ 1091 (shortest path in Binary matrix)
④ 286 (walls and gate (locked).
⑤ Note:- Bfs में marking करते समय first

Lecture-4. (11/oct/2020). [on on bfs].

1. 994 // Rotting oranges.
2. 296. (best meeting point (2 methods).
3. 815 Bus Route (Hin).

Lec-5 (14/10/2020) (directed acyclic graphs).

1. Topological sort
   - ① Using dfs (This fail in cycle).
   - ② Using dfs (impromised)
   - ③ Using Bfs (kahn's algorith).
2. 207 (Course Schedule).
3. 210 (Schedule - II).

Lect-6. (16/10/2020). (directed graphs)

1. Strongly connected component
2. Longest ring path in matrix (using khan's algo.) // 329
3. Kosaraju algo (Find all distinct comp).
4. Bus Route (815) based on ~~locate~~
5. 694 (distinct island count).

Lect-7 (18/10/2020)

1. unionfind. Algorithm (very Imp).
2. 684 // find Redundant connection.
3. lexicographically smallest equivalent string
4. // 200 find num of island (teach U find in matrix)

Lecture-8. [19-10-2020).

① 305 // count islands
② 839 // Similar string count } using union find
③ 990
④ 1135
⑤ Kruskal Algorithm.
⑥ 1168 // water supply cost
⑦ Mr. president on hackerearth.

Lecture-9.

① Prims algorithm
   → our implementation.
   → standard implementation.

② dijekstra
   → implementation standard.

③ Bellmanford
   → our implementation (extra space).
   → standard implementation.

④ On on prims and dijekstra.
   ① 787
   ② 743.

APCO
Date: / /

lect-1

graph 5/10/2020

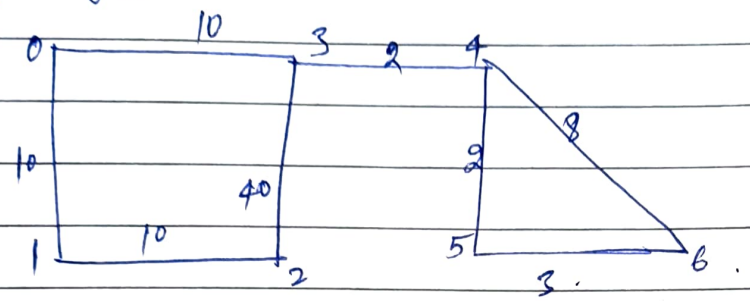
① const → display.

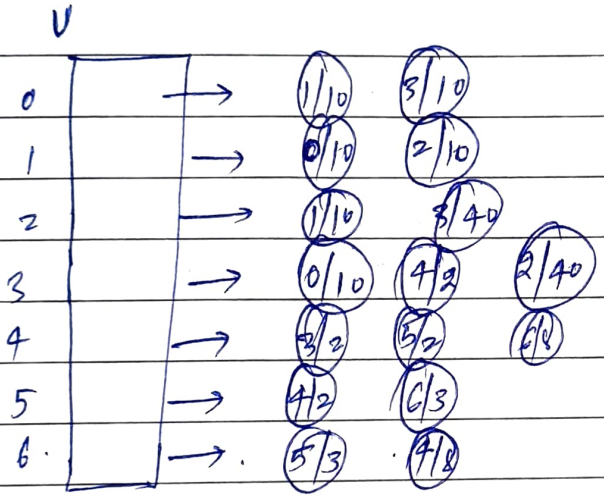
② Remove edge.
③ Remove vertex (only remove edge).
④ has path (b/w s, d).
⑤ Print all path.
⑥ Max wt / Min wt

APCO
Date: / /

① graph Representation.

M1 adjacency list





t: O(9XE)

M-2 Matrix. v to (src) destination v to.

↓	0	1	2	3	4	5	6.
0		10		10			
1	10		40				
2				40			
3	10		40		2		
4				2		2	8
5					2		3
6				3	8		

t: O(nxn)

graph (Main topics)

1. DFS
2. BFS
3. UF

(bigktra, krushkal, prims, floyed warshal) depen ds on ~~dsf~~ dfs

APCO
Date: / /

construction of graph:

```
Public static class Edge {
    int v = 0;
    int w = 0;
    Public Edge (int v, int w) {
        this.v = v;
        this.w = w;
    }

Static int N = 7;
Static AL<Edge> [] graph = new AL[N];

Public static void addEdge (int u, int v, int w) {
    graph[u].add(new Edge(v,w));
    graph[v].add(new Edge(u,w));
}

Public static void constructGraph () {
    for(int i=0; i<N; i++)
        graph[i] = new AL<Edge>();

    addEdge(0,1,10);
    addEdge(0,3,10);
    add Edge(1,2,10);
    add Edge(2,3,40);
    addEdge(3,4,2);
    addEdge(4,5,2);
    add Edge(5,6,3);
    addEdge(4,6,8);
}
```

eg

```
int [] name = new int[];
Type [] name = Type [] ;
                  size
```

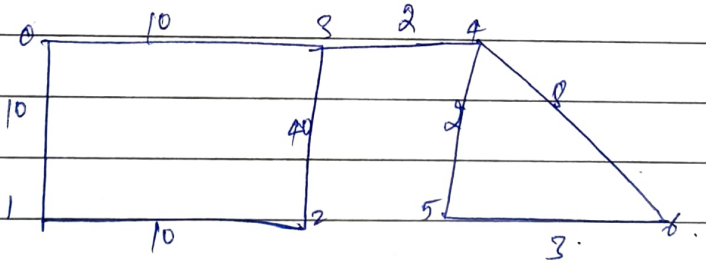
APCO
Date: / /

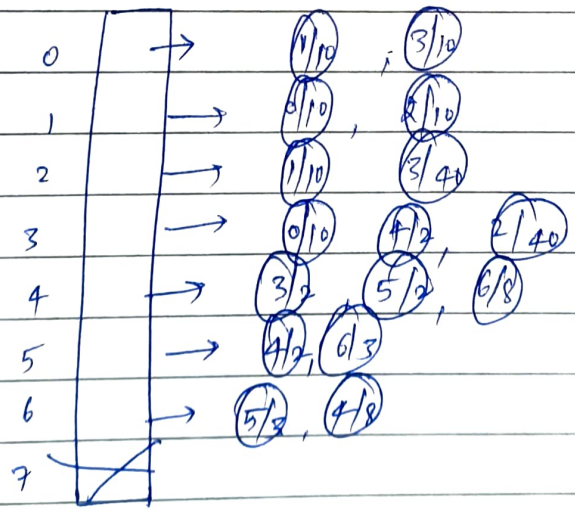
Display of graph (using ~~DFS~~)

```
Public static void display() {
    for (int i=0; i<N; i++) {
        System.out.println(i + "->");
        for (Edge e : graph[i]) {
            System.out.print("(" + e.v + "),");
        }
        System.out.println();
    }
}
```

Complexity (2XE)

because vtx is added at src and once at destination twice.





695 → Perimeter
130 → max area surrounded Region.

APCO
Date: / /

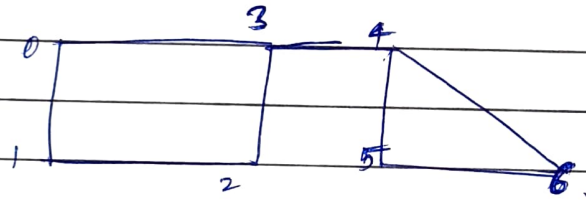
As total 16 Edge. (2×E)

2. Remove Edge:

DFS (Algo)

① mark.
② For all unvisited nbrs.
(2-1) Call for dfs.(nbrs)

what ?



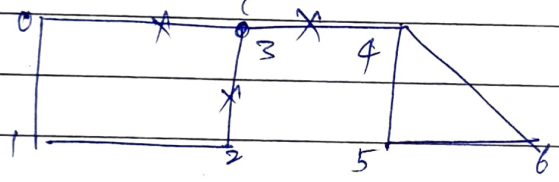
⇒ (Remove (3,4) edge. then have to remove it from vtx 3 and vtx 4 both).

```
Public static int searchVtx( int u, int v) {
    for( int i=0; i< graph[u].size(); i++) {
        Edge e = graph[u].get(i);
        if( e.v == v) return i;
    }
}
```

APCO
Date: / /

```
Public static void removeEdge(int u, int v){
    int l1 = searchVtx(u,v);
    graph[u].remove(l1);

    int l2 = searchVtx(v,u);
    graph[v].remove(l2);

}
```

3. On Remove Vtx (only Remove all Edges). (But still Remain in array)
what.



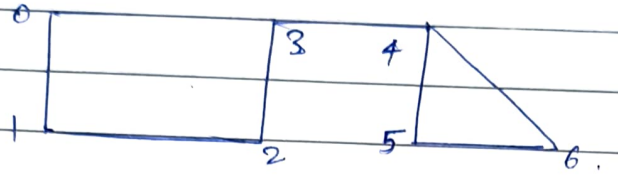
How:

loop start from end
if start from then
there will be
shift o(n) time

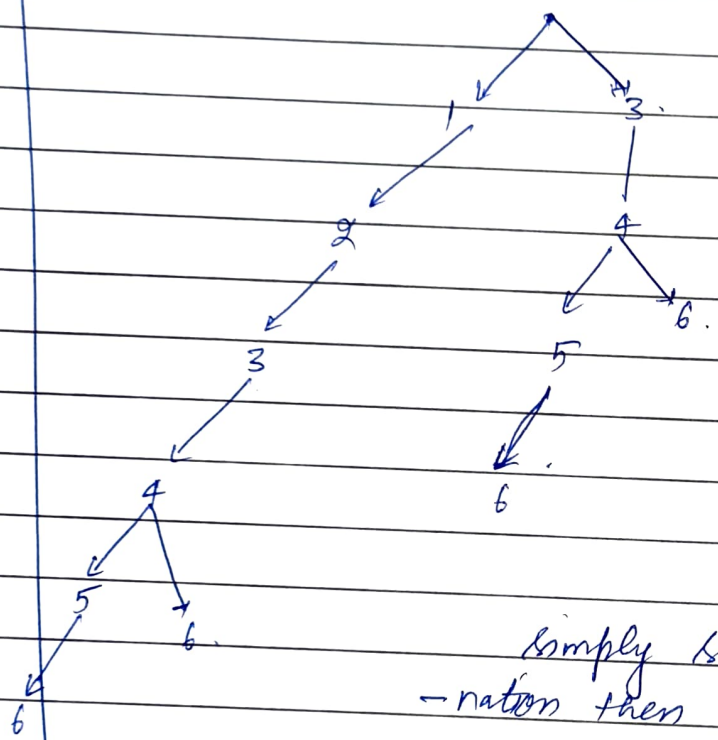
```
Public static void removeVtx(int u){
    for(int i = graph[u].size()-1; i>=0; i--){
        Edge e = graph[u].get(i);
        removeEdge(u, e.v)

    }
}
```

APCO
Date: / /

4. Qu. Has path.



What ? Simply search for path b/w src and distination.



Simply search for path ; if ~~search~~ each destination then return true.

APCO
Date: / /

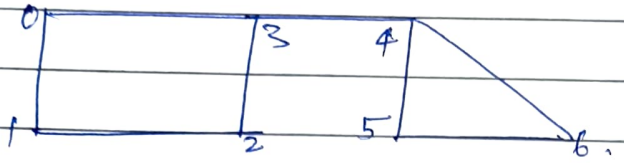
// code.

```
Public static boolean hasPath(int src, int dest, boolean[] vis){
    if(src == dest) return true;

    vis[src] = true;
    boolean res = false;
    for(Edge e : graph[src]){
        if(!vis[e.v])
            res = res || hasPath(e.v, dest, vis)
    }
    return res;
}
```

APCO
Date: / /

⑤ Print all path.

What ? Print and count all path from src to destination.

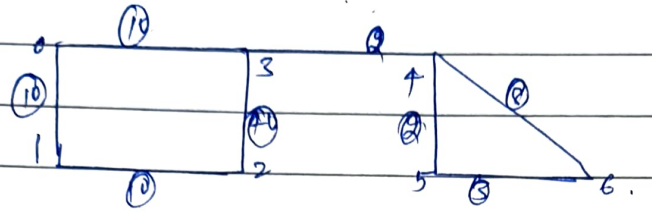



```
Public static int allPath(int src, int dest, boolean[] vis,
    string psf, int wsf) {
    if(src == dest) {
        sysoln(psf + src + "@" + wsf);
        return 1;
    }
    vis[src] = true;
    int count = 0;
    for(Edge e : graph[src]) {
        if(!vis[e.v]) {
            count += allPath(e.v, dest, vis, psf + src
                + "" + wsf + e.w);
    }
    vis[src] = false;
    return count;
}
```

APCO
Date: / /

6.Ou Max / Min weight :-

What ?



Print path having max/ min weight from 'src' to destination.

How-?

Make a pair class having, Path, wt, isdestHit, call for each edge and compare path after each dfs call.

// Code :-

```
Public static class Pair {
    String path = "";
    int weight = 0;
    boolean isDestHit = false;

    Pair (String path, int weight, boolean isDestHit) {
        this.path = path;
        this.weight = weight;
        this.isDestHit = isDestHit;
    }
}
```

APCO
Date: / /

```
public static pair maxWeightPath(int src, int dest, boolean[] vis){
    if(src == dest){
        return new pair(src + "", 0, true);
    }
    vis[src] = true;
    pair myAns = new pair("", 0, false);
    for(edge e: graph[src]){
        if(!vis[e.v]){
            pair recAns = maxWeightPath(
                e.v, dest, vis);
            if(recAns.isDestHit && recAns.weight + e.w
                > myAns.weight){
                myAns.weight = recAns.weight + e.w;
                myAns.path = src + " " + recAns.path;
                myAns.isDestHit = true;
            }
        }
    }
    vis[src] = false;
    return myAns;
}
```

APCO
Date: / /

7. Min weight

APCC

Date: / /

Lecture - 2

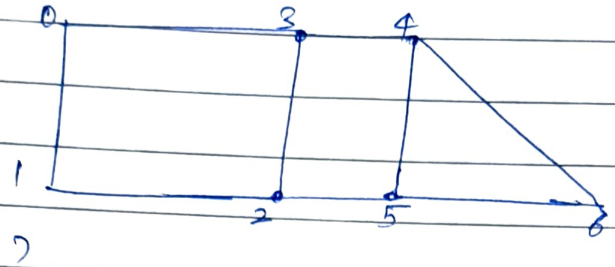
graphs (7/oct/2020).

agenda: (dfs based on) and (L-1) also.

1. Hamiltonian path and cycle
2. No. of islands (#200).
3. Max. area of island (#695).
4. #463. #max perimeter of island.
5. 130 #Leetcode #surrounded Region.
6. 694. (H.w). No. of distinct islands.

APCO
Date: / /

① Hamiltonian path and Cycle.



What ?

Hamiltonian path :- Path of src to destination such that all of its ~~node~~ vtx get covered.

eg if n vtx then
path should have (n-1) vtx.
path : 0 1 2 3 4 5 6.

Hamiltonian cycle :-
H. Path from src to dest such that there is edge b/w src and destination.

eg. 0 3 4 6 5 2 1

How. Apply ~~bfs~~. and make count of vtx print only those path which have (n-1) vtx.

APCO
Date: / /

Note:- Do dry Run and mid of it whatever ds you think you should use them in your code that's it.

// Code:

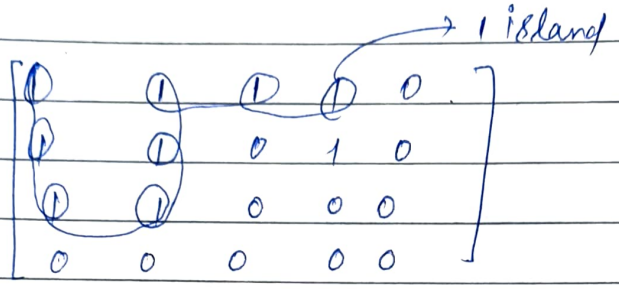
```
public static int hamintonianpath(int src, int osrc,
    boolean[] vis, int edgeCount, string psf){

    if (edgeCount == N-1){
        psf += src;
        int idx = searchVtx(src, osrc);
        if(idx != -1) sysoln("Cycle: "+psf);
        else
            sysoln("path:"+psf);

        return 1;
    }

    vis[src] = true;
    int count = 0;
    for(Edge e: graph[src]){
        if(!vis[e.v]){
            count += fn(e.v, osrc, vis,
                edgeCount+1, psf+src+"");
    }
    vis[src] = false;
    return count;
}
```

APCO
Date: / /

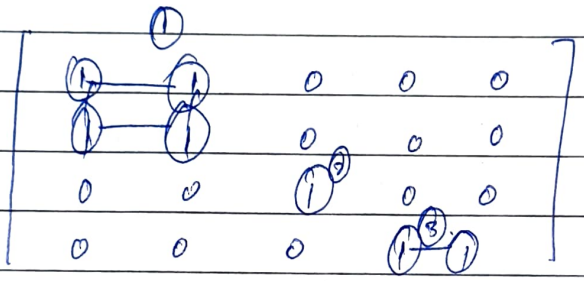
② Leet Code - (200).

No. of island count

What



1 = land
0 = water.



3 island

How :- Here we make a direction array which gives movement from a coordinate. eg.

```
Public int numIslands(char[][] grid){
    int[][] dir = {{0,1},{0,-1},{1,0},{-1,0}};
    int count = 0;
    for(int i=0; i< grid.length; i++){
        for(int j=0; j < grid[0].length; j++){
            if(grid[i][j] == '1'){
                numIslandsDFS(grid, i, j, dir);
                count++;
            }
        }
    }
    return count;
}
```

APCO
Date: / /

whenever we go to any vtx check whether '1' or not then if not then put = 1

```
Public Void numIslandsDFS(char[][] grid, int r,
    int c, int[][] dir) {
    grid[r][c] = '0';
    for(int d = 0; d < dir.length; d++) {
        int x = r + dir[d][0];
        int y = c + dir[d][1];

        if(x >= 0 && y >= 0 && x < grid.length &&
            y < grid[0].length && grid[x][y] == '1') {

            numIslands(grid, x, y, dir);
        }
    }
}
```

APCO
Date: / /

(3) Max Area Of Island. (695).

What ? Here we have to return max Area Island.

① Here we count size of each island.
② Compare island size with max. of island count variable.

→ just add these thing baki same as previous.

// Code.

```
Public int maxAreaOfIslandDFS(int[][] grid, int r,
    int c, int dir[][]) {
        grid[r][c] = 0;
        int count = 0;
        for(int d=0; d < dir.length; d++){
            int x = r + dir[d][0];
            int y = c + dir[d][1];
            if( x >=0 && y >=0 && x < grid.length
                && y < grid[0].length && grid[x][y]
                == 1 ) {
                count += fn(grid, x, y, dir);
            }
        }
        return count+1;
}
```

APCO
Date: / /

② fn:

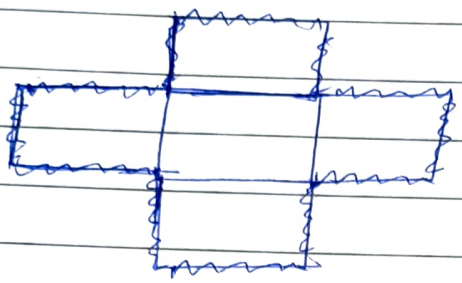
```
Public int maxAreaofIsland(int[][] grid) {
    int[][] dir = {{0,1},{0,-1},{1,0},{-1,0}};
    int maxArea = 0;
    for(int i=0; i<grid.length; i++){
        for(int j=0; j<grid[0].length; j++){
            if(grid[i][j]==1){
                maxArea = Math.max(maxArea, mAfs(grid,i,j,dir));
            }
        }
    }
    return maxArea;
}
```

M-2 Try with Bfs also.

APCO
Date: / /

4Q) 463 island perimeter.

what.



Perimeter = 20 - 4*2

= 12

How?

① when encounter '1' count 01 and call all 4dirn and call its nbr's where one exist and do for all nbrs having '1'.

Ans = 4 * one'Count - nbr's count

APCO
Date: / /

Code:

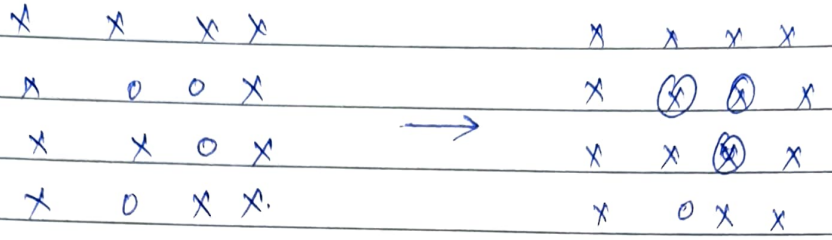
```
// Public int islandPerimeter(int[][] grid){
    int[][] dir = {{0,1},{0,-1},{1,0},{-1,0}};

    int countOne = 0;
    int countNbrs = 0;

    for(int i=0; i< grid.length; i++){
        for(int j=0; j < grid[0].length; j++){
            if(grid[i][j] == 1){
                countOne++;
                for(int d=0; d<4; d++){
                    int r = i + dir[d][0];
                    int c = j + dir[d][1];

                    if(r>=0 && c>=0 && r< grid.length; r< grid[0].length;
                    && grid[r][c] == 1){
                        nbr++;
                    }
                }
            }
        }
    }
    return countOne - nbrs;
}
```

app-2 also try using ← ↓ direction movement

4* one count - 2* nbrs ans होगा.

APCO
Date: / /

⑤ 130 Surrounded Region.



have to flip O → X.
But only flip those O
① should not be on border.
② 'O' connected with border 'O' should also not be flipped.

How: ① mark all boundry 'O' and connected 'O' with boundry with any other symbol say '$'.
② now move on grid and mark every i,j with 'O' = X.
③ now Unmark '$' = 'O'

// code:

```
Public void surroundedRegions(char[][] grid, int r,
    int c, int[][] dir){
    grid[r][c] = '$';

    for(int d=0; d<4; d++){
        int x = r + dir[d][0];
        int y = c + dir[d][1];
```

$\frac{120}{100} = 1.20$

APCO
Date: / /

```
        if( x>=0 && y>=0 && x<grid.length && y<grid[0].length
            && grid[x][y] == 'O')
            SurroundedRegionDFS(grid, x, y, dir);

        }
    } //for

} //fn.

Public void solve(char[][] grid){
    int[][] dir = {{0,1},{0,-1},{1,0},{-1,0}};

    for(int i=0; i<grid.length; i++){
        for(int y=0; y<grid[0].length; y++){
            if(i==0 || y==0 || i==grid.length-1 ||
               y==grid.length-1){
                if(grid[i][j] == 'O'){
                    SurroundedRegionDFS(grid, i, j, dir);
                }
            }
        }
    }
    for(int i=0; i<grid.length; i++){
        for(int y=0; y<grid[0].length; j++){
            if(grid[i][j] == 'O') grid[i][j] = 'X';
            else if(grid[i][j] == '$')
                grid[i][y] = 'O';
        }
    }
} //fn end.
```

APCO
Date: / /

6 Oct

GQ 4 (No. of distinct island).

APCO Date: 9/10/2024

Lect-3
(graph)

Agenda:

1. BFS traversal.
   - T-1 Normal BFS. (just put and remove).
   - T-2 BFS with null pointer. (Put null at end of level).
   - T-3 BFS with cycle. (किसी भी vtx का src से shortest dist keep in ar).
   - T-4 BFS without cycle. (same as BFS 3 just mark before put).
2. Qu. 785 (is graph Bipartite).
3. Qu. 1091 ~~286~~ | Shortest path in matrix.
4. Qu. 286 || Boundry and wall.

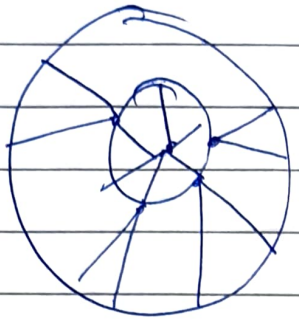
Note BFS में marking पहले करने से थोड़ा fast चलता है।

APCO
Date: / /

① Bfs.

Bfs gives the shortest path.

becoz we are moving unit by unit



→ It is cycle detection also and do not tell how many cycles are there.

→ Bfs always gives better time Complexity then dfs.

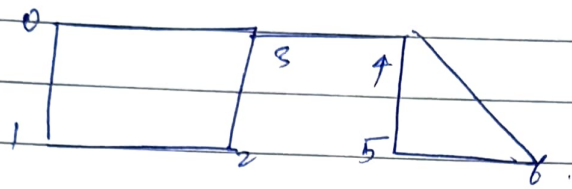
Algorithm of Bfs-

① Make a queue.
② que. ← src.
③ while (que. size () != 0).
  3.1 vtx ← que.
  3.2 mark vtx.
  3.3 for all unvisited neighbours.
    3.3.1 insert into que ← nbrs.

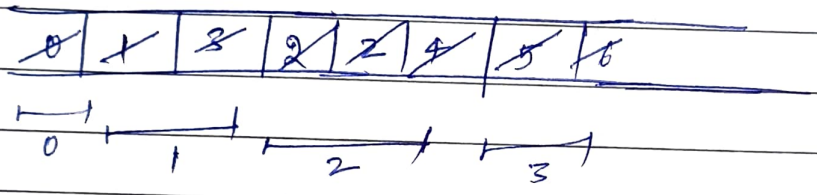
APCO
Date: / /

① BFS 01 (on detect cycle).

dry run.



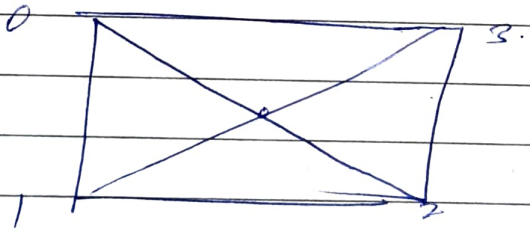
V	D.
0	T
1	T
2	T
3	T.
4	T
5	T
6.	T.



→ 9t is just normal bfs.

→ 9t detects cycle but doesn't give us cycle count.

eg.



Here count comes out = 4. cycle.
but totally 9 cycle present so not good.

→ cycle count not in syllabus.

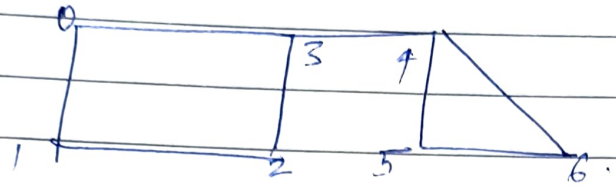
APCO
Date: / /

// Code -

```
public void BFS_01 (int src, boolean[] vis){
    ll<Integer> que = new LL<>();
    que.addLast(src);
    boolean cycle = false;

    while(que.size() != 0){
        int vtx = que.removeFirst();
        if(vis[vtx])
            cycle = true;
            continue;
        }
        vis[vtx] = true;
        for(Edge e: graph[vtx]){
            if(!vis[e.v]){
                que.addLast(e.v)
            }//if
        }//for
    }//while
}//fn
```

APCO
Date: / /

20m BFS_02 (On find shortest distance of node 6.



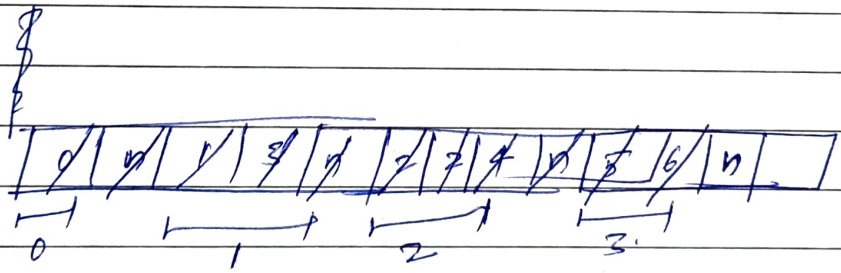
M-1 maintain an pair class of vtx and level and when put other vtx ges level by and put with value.

eg (0/0).

and so on.

M-2 Maintain a null pointer and count level++ when we encounter (delimintor) null.

eg.



Note :- to get longest path we use 'dfs' and update path in recursion.
Bfs donot gurantee. longest path.

APCO
Date: / /

M-3 ⇒ Calculate size and run inner while loop till size and count level++, at each outer while loop.

M-4 // Code - best.

```
Public static void Bfs_02(int src, boolean [] vis){
    LL<Integer> que = new LL<>();
    que.addLast(src);
    que.addLast(null);

    boolean cycle = false;
    int level = 0;
    while(que.size() != 0){
        int vtx = que.removeFirst();
        if(vis[vtx]){
            cycle = true;
            continue;
        }

        if(vtx == dest){
            sysout(level);
        }
        vis[vtx] = true;

        for(Edge e : graph[vtx]){
            if(!vis[e.v]){
                que.addLast(e.v);
            }
        }
    }
```

APCO
Date: / /

```
if( que.getFirst() == null ){
    level++;
    que.addLast( que.removeFirst() );
}
}// while
}// fn
```

APCO Date: / /

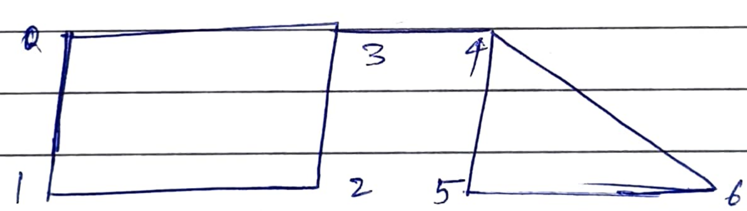
(Concept, dryrun, code, कोई नयी problem मिले atleast 5 m copypen पे socho)

3.20 **BFS_03**:
(every Edge is at how many minimum distance).

Approach:
Here we do same a pre app. just maintain arr of size(N) and put value of level at each iteration (
∵ level gives min dist from src node to given node).

Note :- Code लिखना 5 पैसे का काम है (4L-5L).
main काम है
Thinking and dry run on copy pen (12-15) L.

dry Run:-



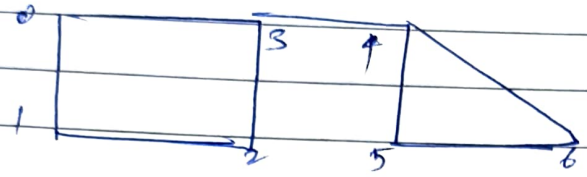
Here all thing remain same except put

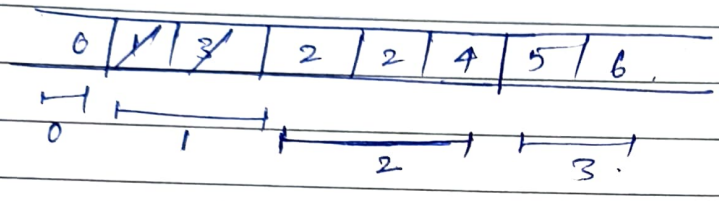
dis[v+x] = level;
after Continue वाली line.

and we add here only because if से पहले add करे तो level might change.

APCO
Date: / /

eg





(Here do not change but (might change in some cases.)

Note- BFS_03_WithCycle(int src, boolean[] vis){

{ // same.
}.

int dis[] = new int[N];.

while (que.size()!=0) {
{ //
while() {

// - - -

dist[rvto] = level;.

// same.

}

}

APCO
Date: / /

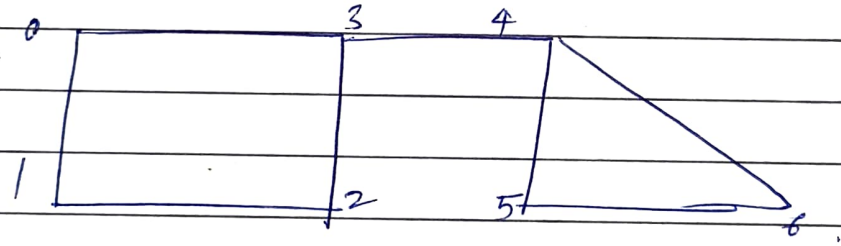
Note. BFS_03: solve nearly all cycle related problem.

4 Qn BFS 4 बिना cycle वाला

→ marking and putting element in queue at same time.

good → Complexity wise.
bad → do not detect cycle.

dry run:



0 ✓
1 ✓
2 ✓
3 ✓
4 ✓
5 ✓
6 ✓

0	1	3	2	4	5	6

APCO
Date: / /

// Code:

diff from previous code is that remove cycle detecting lines and put vis[e.v] = true for while putting element(vtx) into queue.

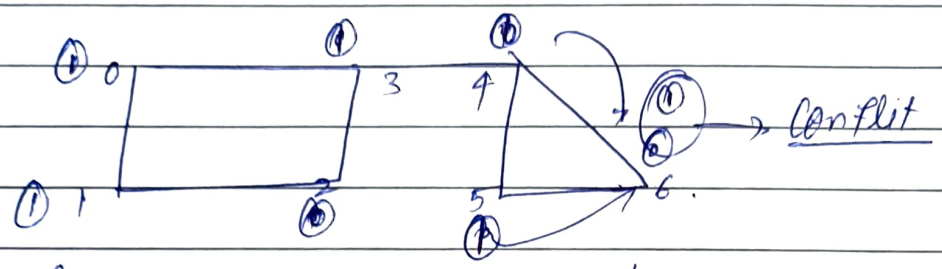
```
bfs_of_acyclic_graph(int src, boolean[] vis){

    while(){
        ---
        while(){

            for(edge e : graph[vtx]){
                if(!vis[e.v]){
(diff)              // vis[e.v] = true;
                    que.addLast(e.v);
                }
            }
```

APCO
Date: / /

Que Tes (graph is Bipartite or not).

Means - if we colour alternatively then any one u to don't contradict of two different colour.

eg =>



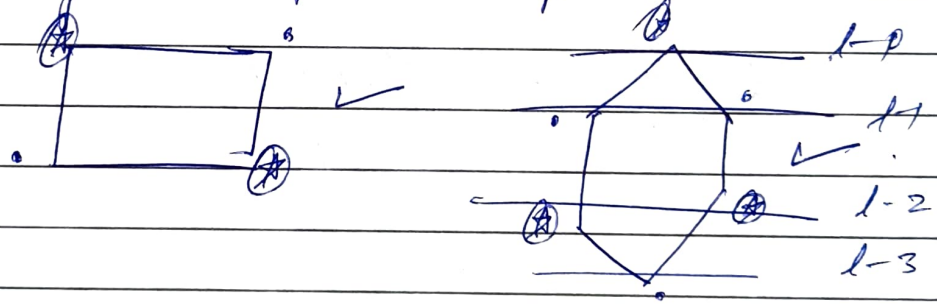
0 → Red

1 → green.

(so not bipartite)

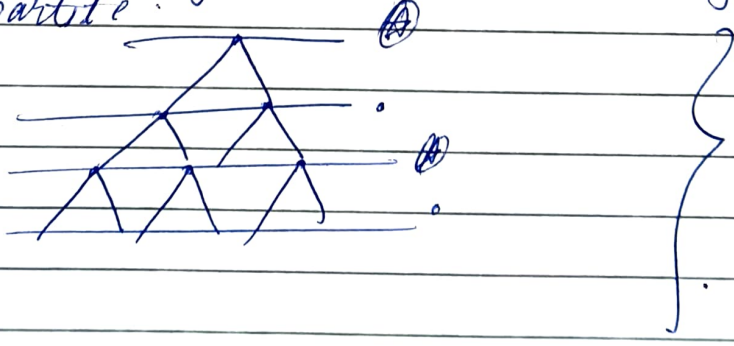
Condition to be bipartite

① Even edge cycle is bipartite except '2'.



Imp concept of Bipartite.

② if graph is not cyclic then bipartite. (0 cycle).
tree is bipartite.

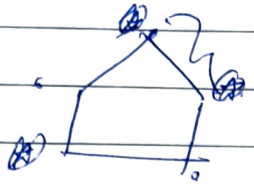


① O(N)
independent of N → O(N) complexity होगी

② for every element loop N time चलेगा तो then complexity of n^2 होगी

③ Not need 2 loop inside loop = $O(n^2)$ complexity

③ odd edge length graph is not bipartite.

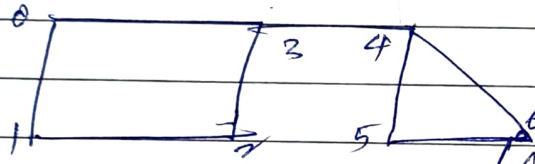


Conflict.

X.

④ if even and odd both length then not bipartite

eg

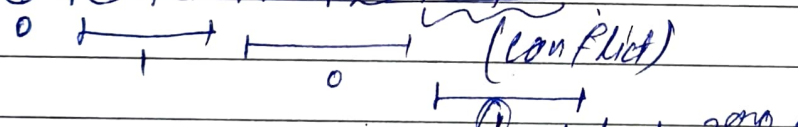


(Conflict point)

0	-1
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1

0 → Red.

1 → green.



① but get zero also conflict created.

must see how graph को Loop-तोड़ रहे हैं on Date: 1 . 1

Imp

(1)

```
public boolean isBipartite(int[][] graph, int[] markColor, int src){
        LL<Integer> que = new LL<>();
        que.addLast(src);
        int color = 0;
        while(que.size() != 0){
            int size = que.size();
            while(size-- > 0){
                int vtx = que.removeFirst();
                if(markedColor[vtx] != -1){
                    if(markedColor[vtx] != color){ //
                    (// phle red tha ab white se
                          mark krne aa gya)
                    return false;
                    }
                    continue;
                }

            markedColor[vtx] = color;
            for(int e : graph[vtx]){
                if(markedColor[e] == -1){
                        que.addLast(e);
                }
            }
        }
            color = (color+1) % 2
        } // outer while
        return true;
}
```

APCO
Date: / /

⑧ Public boolean isBipartite(int[][] graph){
int N = graph.length;
int[] markedColor = new int[N];
Arrays.fill(markedColor, -1);

boolean ans = true;
for(int i=0; i<N; i++){
if(markedColor[i]==-1){
ans = isBipartite(graph, markedColor, i);
if(!ans) return false;
}
}
return true;
}

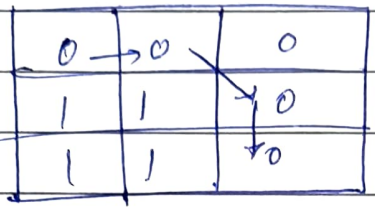
idx = x*m+c. *

APCO
Date: / /

Que 1091 (shortest path in Binary matrix).

what NxN grid 0 → empty, 1 → blocked.
8 directional movement allowed, find shortest path from src to destination.

eg.



4 steps req to reach dest.

How :- do using bfs as bfs move in radius to. it will give the shortest distance. put only those nodes in que which are not visited.

APCO
Date: / /

// Code:

```
Public int spBM(int[][] grid){
    if(grid.length==0 || grid[0].length==0) return -1;
    int n= grid.length;
    int m= grid.length;
    if(grid[0][0]==1 || grid[n-1][m-1]==1) return -1;
    LL<Integer> que = new LL<>();
    que.addLast(0*m+0);
    grid[0][0]=1;   // marked as visited.
    int[][]dir ={{1,0},{-1,0},{0,1},{0,-1},{1,1},{-1,-1},
                 {-1,1},{1,-1}};
    int level=1;

    while(que.size()!=0){
        int size = que.size();
        while(size-->0){
            int idx = que.removeFirst();
            int r = idx/m;
            int c = idx%m;
            if(r==n-1 && c==m-1) return level;

            for(int d=0; d<8; d++){
                int x = r+dir[d][0];
                int y = c+dir[d][1];
                if(x>=0 && y>=0 && x<n && y<m
                    && grid[x][y]==0){
                    grid[x][y]=1;
                    que.addLast(x*m+y);
                }
            }
        } // interview
        level++;
    } // outer while
    return -1;
}
```

APCO
Date: / /

Qu 286 ~~Boarding walls~~ walls and gates (locked)

(whenever we have to find shortest kind then use bfs).

What :-

2-d matrix. , 0 → gate, -1 ⇒ wall, ∞ = free space

fill each empty room with the distance to its nearest gate, if impossible to reach gate fill (∞).

eg.

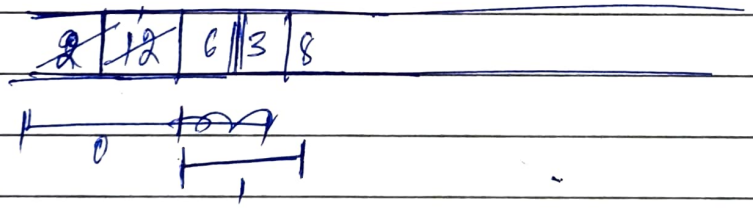
∞	-1	0	∞
∞	∞	∞	-1
∞	-1	∞	-1
0	-1	∞	∞

⇒

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

How-

Put all zero in que and apply bfs on que. and put all index with ∞. and mark visited.



APCO
Date: / /

Code:

```
Public void wallsAndGate(int[][] rooms) {
    if(rooms.length==0 || rooms[0].length==0) return;
    int n = rooms.length;
    int m = rooms[0].length;
    LL<Integer> que = new LL<>();
    boolean[][] vis = new boolean[n][m];
    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            if(rooms[i][j]==0){
                que.addLast(i*m+j);
                vis[i][j] = true;
            }
        }
    }
    int level=0;
    int[][] dir = {{0,1},{0,-1},{1,0},{-1,0}};
    while(que.size()!=0){
        int size = que.size();
        while(size-->0){
            int idx = que.removeFirst();
            int r = idx/m;
            int c = idx%m;
            room[r][c] = level;
            for(int d=0; d<4; d++){
                int x = r+dir[d][0];
                int y = c+dir[d][1];
                if(x>=0 && y>=0 && x<n &&
                    y<m && !vis[x][y] &&
                    room[x][y]!=-1){
                    vis[x][y] = true;
                    que.addLast(x*m+y);
                }
            }
        }
        level++;
    }
}
```

APCO
Date: / /

Lecture-4. (11/oct/2020.)
(graph:)

Agenda: (हर एक में लगता है BFS है bs किस तरह से लगाते है that will matter).

① 994. // Rotting oranges.
② 296 // Best meeting point (locked) (M-1, M-2).
③ 815 // Bus Route.

APCO
Date: / /

① Que 994 // Rotting oranges

What ?

0 → empty cell, 1 → Fresh orange, 2 → rotten orange:
every minute Fresh orange adyacent to rotten get rotte. Find min no. of minute take to rotte all fresh orange.
if not possible return -1;

2	1	1
1	1	0
0	1	1

apply bfs.

algo:

how :- ① Count total fresh oranges present if (0) return 0.
② Put all cell having '2' into que and assign level 0.
② Now Remove elements from que and dment fresh count by -1.
③ when (fresh count == 0) return level.

APCO
Date: / /

994 Code.

```
Public int rottingOranges(int[][] grid){
    if(grid.length==0 || grid[0].length==0) return 0;
    int n = grid.length;
    int m = grid[0].length;
    int[][] dir = {{0,1},{1,0},{0,-1},{-1,0}};
    LinkedList<Integer> que = new LL<>();
    int freshOrange =0;
    for(int i=0; i<n ;i++){
        for(int j=0; j<m; j++){
            if(grid[i][j]==2) que.addLast(i*m+j);
            if(grid[i][j]==1) freshOranges++;
        }
    }
    if(freshOranges==0) return 0;
    int time =0;
    while(que.size()!=0){
        int size = que.size();
        while(size-->0){
            int idx = que.removeFirst();
            int r = idx/m;
            int c = idx%m;
            for(int d=0; d<4; d++){
                int x = r+dir[d][0];
                int y = c+dir[d][1];
                if(x>=0 && y>=0 && x<n && y<m &&
                   grid[x][y]==1){
                    grid[x][y]=2;
                    freshOranges--;
                    que.addLast(x*m+y);
                    if(freshOranges==0) return time+1;
                }
            }
        }
        time++;
    }
    return -1;
}
```

APCO
Date: / /

2 Qu | 296 // Best meeting point (using BFS) // not locked on LC.

My-1

What? A grp of people want to meet and minimize total distance travel. They can meet at any point.
given grid of '0' and '1' ⟹ 1 → is home of people.
Manhattan dist(p1, p2) = |p2.x − p1.x| + |p2.y − p1.y|

eg

1	0	0	0	1
0	0	0	0	0
0	0	1	0	0

eg

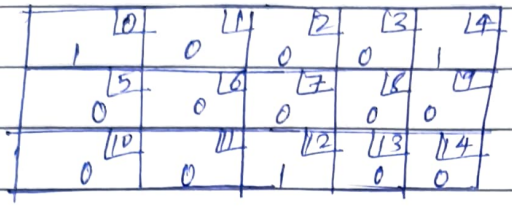
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

How's-

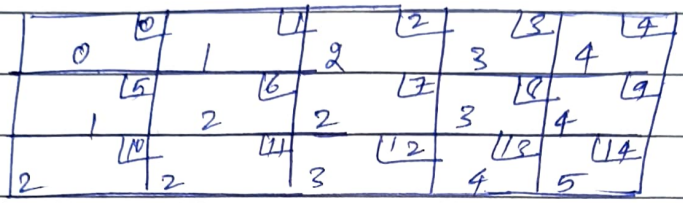
1. make an array of (2-D) count/dist and visited
2. loop through the whole grid. and apply bfs. when we find "1" put into que.
3. put the level as a distance in ~~distance~~ count grid.
4. do this for all one.
5. ~~At~~ at last find the cell of count/dist having lowest value and that will be our answer.

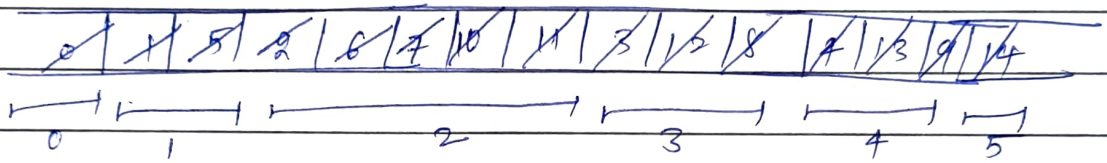
APCO
Date: / /

dry Run:



dist count for '1' at '0' index.





Note:

→ also maintain visited arr of 2D to keep track which cell have been visited.

→ do same for as many 1 present from that cell (r, c)

→ if meeting point is that cell of 1 then its level (i.e) 0 will be add to dist matrix.

APCO
Date: / /

M-2 ~~using manhatten distance formula~~ all

// Code [This code gives the when cell have 1.

```
Public void Bfs(int i, int j, int[][] grid, int[][] dir,
        int[][] dis){
    int n = grid.length();
    int m = grid[0].length;
    boolean[][] vis = new boolean[n][m];
    LinkedList<Integer> que = new LL<>();
    que.addLast(i*m+j);
    vis[i][j] = true;
    int level=0;
    while(que.size()!=0){
        int size = que.size();
        while(size-->0){
            int idx = que.removeFirst();
            int r = idx/m;
            int c = idx%m;
            dist[r][c] += level;
            for(int d=0; d<4; d++){
                int x = r+dis[d][0];
                int y = c+dis[d][1];
                if(x>=0 && y>=0 && x<n
                    && y<m && !vis[x][y]){
                    vis[x][y] = true;
                    que.addLast(x*m+y);
                }
            }
        } // in while
        level++;
    } // outer while
} // fn
```

APCO
Date: / /

```
Public int minTotalDist(int[][] grid){
    if(grid.length==0 || grid[0].length==0) return 0;
    int n = grid.length;
    int m = grid[0].length;
    int[][] dir = {{0,1},{0,-1},{1,0},{-1,0}};
    int[][] dis = new int[n][m];

    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            if(grid[i][j]==1){
                BFS(i, j, grid, dir, dis);
            }
        }
    }

    int ans = (int)1e8;
    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            ans = Math.min(ans, dis[i][j]);
        }
    }
    return ans;
}
```

APCO
Date: / /

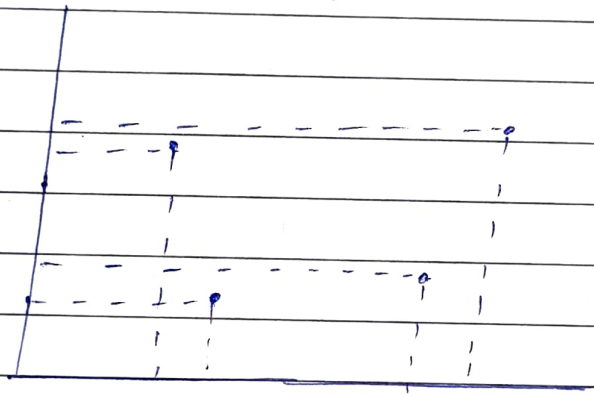
~~2 Oct~~ ~~815 // Bus Routes~~.

M-2. Previous app. gave the when all '1' are in grid 80. apply other app. of manhattan formula.

Median :- odd element हो तो बीच का element और even हो तो $\frac{x_i + x_{i+1}}{2}$

How - Take reflection of all '1' on x and y axis respectively

eg



(1) Refl^n on x axis will be column no.
(2) " " y axis " " row "

$dx = |x' - x|$
$dy = |y' - y|$

eg.

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

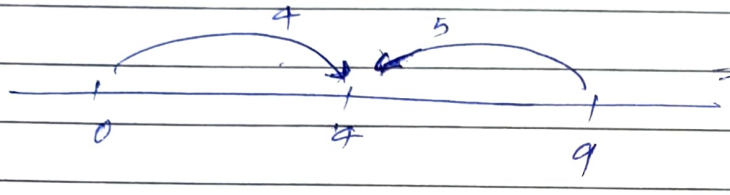
x : [0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4]
y : [0, 1, 2, ~~3, 4~~, 0, 1, 2, ~~3, 4~~, 0, 1, ~~2, 3, 4~~]
0, 1, 2, 0, 1, 2)

APCO
Date: / /

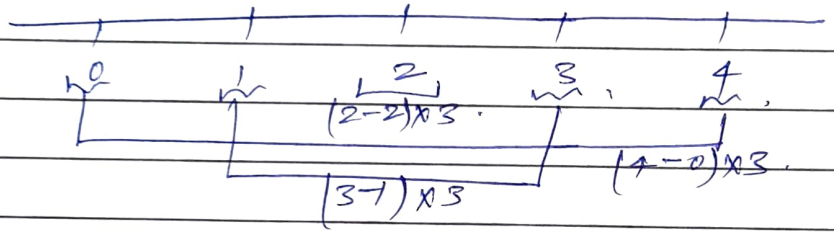
At one time we calculate either 'x' distance or 'y'

① We know to travel one end to other have to take 'n' unit if start from zero.

② Consider melting point in middle and calculate distance for 'x' and 'y' respectively.

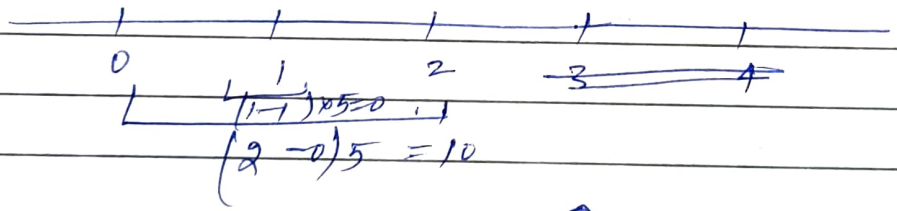


= 9 steps.



total dist in x = 12 + 6 = 18.

sort y points



total = 18 + 10 = 28. ✓

APCO
Date: / /

// Code.

```
Public int getInter(Arraylist<Integer> list){
    Collection.Sort(list);
    int res = 0;
    int i = 0;
    int j = list.size()-1;
    while(i<j){
        res += list.get(j--) - list.get(i++);
    }
    return res;
}

Public int minTotaldistance(int[][] grid){
    if(grid.length==0 || grid[0].length==0) return 0;
    int n = grid.length;
    int m = grid.length;
    Arraylist<Integer> xpoints = new A<>();
    A<> ypoints = new A<>();

    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            if(grid[i][j]==1){
                xpoints.add(i);
                ypoints.add(j);
            }
        }
    }
    return getInter(xpoint) + getInter(ypoints);
}
```

APCO
Date: / /

(8) Ans: 815 Bus Route.

APCO
Date: / /

Lecture-5 (graph). [14-10-2020].

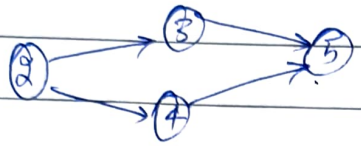
Agenda: [DAG] graph]

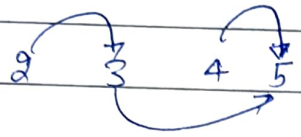
1. Topological sort
   - → using dfs. (This fail in cycle).
   - → using Bfs. (Kahn's algo).
   - → Improving dfs for cycle.
2. 207 // Course schedule.
3. Course schedule - II // 210.

APCO
Date: / /

Topological Sort :- For every directed edge (u,v) u should always come before 'v'!

Ex.





→ Here are the edges are fwd so it is perfect topological sort This method is useful in mcq's.

① **Find topological sort using dfs :-**

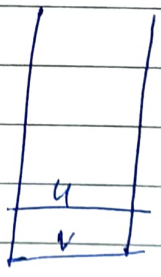
Algo.

① Array list बनाउनी to store answer (basically need stack but we're using AL).

② Boolean की array to check whether node visited / not.

Note :- are not considered in topological order.

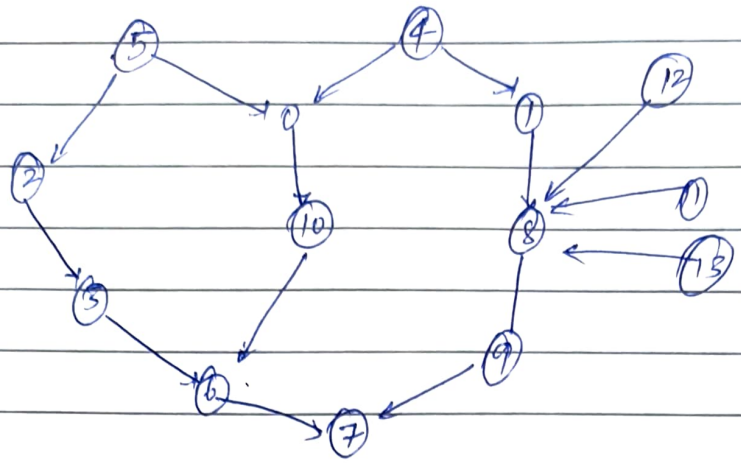
→ First visiting and putting element in queue in reverse ensure. u, will always come before v.



t'o :- u, v

APCO
Date: / /

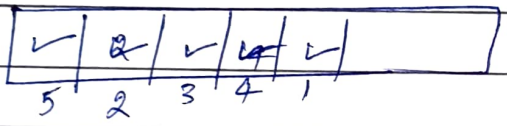
eg.

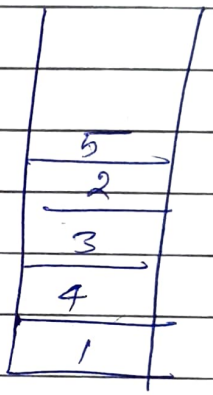


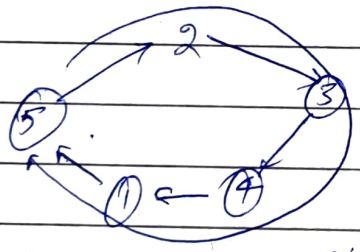
→ To detect cycle if we get same element in stack while call then there will be cycle else not.

→ so, dfs can detect cycle as well.

eg.







→ 5 come again which is visited so contain cycle.
So it is not possible to get its t'sort

→ but can get topological order.
(5, 2, 3, 4, 1)

→ Here addition of element in stack/qL is post order.

APCO
Date: / /

// Code:

①
```
public static void TopoDFS(int src, boolean vis[], ArrayList<Integer> ans){
    vis[src] = true;
    for(Integer e : graph[src]){
        if(!vis[e])
            TopoDFS(e, vis, ans);
    }
    ans.add(src);
}
```

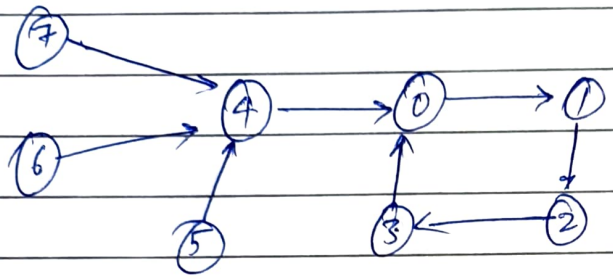
②
```
public static void topoDFS(){
    ArrayList<Integer> ans = new ArrayList<>();
    boolean[] vis = new boolean[N]; // N = no. of vtx.

    for(int i=0; i<N; i++){
        if(!vis[i]){
            topoDFS(i, vis, ans);
        }
    }
}
```

APCO
Date: / /

Drawback of topological dfs:

→ It is not able to detect cycle. So, we do another version of dfs.

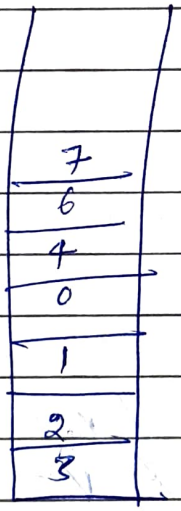
eg where it fails.



boolean:

0 → ✓
1 → ✓
2 → ✓
3 → ✓
4 → ✓
5 → ✓
6 → ✓
7 ↩ ✓

stack:



topo sort :- 7, 6, 5, 4, 0, 1, 2, 3.

contain cycle so, not perfect ti sort

APCO
Date: / /

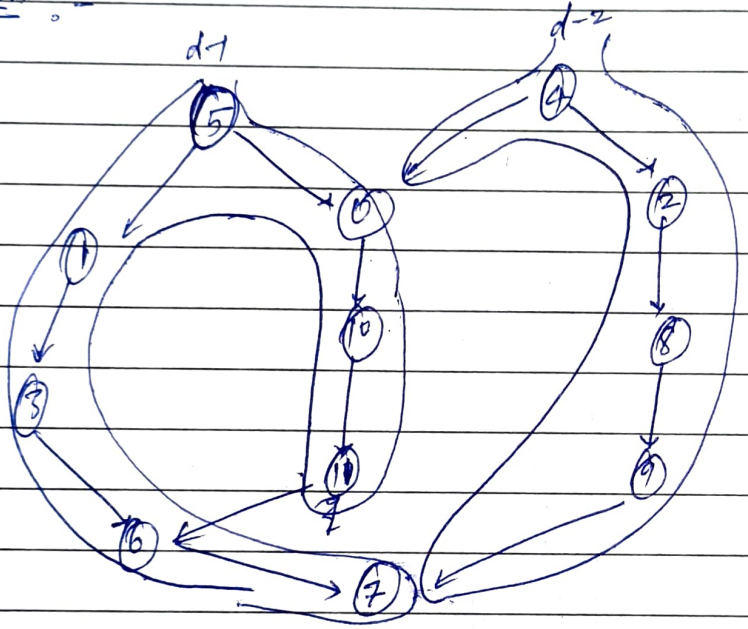
Although this algo not used much but less.

Purpose → cycle detection, topological sort.

Concatenation:-

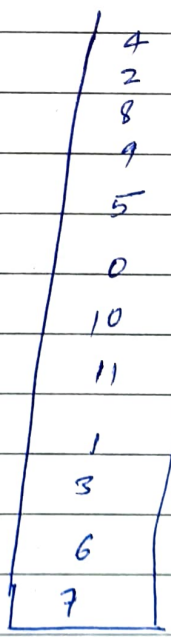
Here we will keep 0 → not visited, 1 → part of path
2 → visited and not part of path.

dry Run:-



call stack

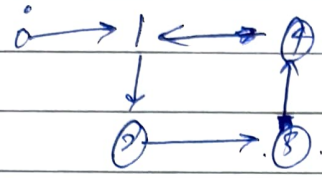
0 → ∅ ✗ 2
1 → ∅ ✓ 2
2 → ∅ ✗ 2
3 → ∅ ✗ 2
4 → ∅ ✓ 2
5 → ∅ ✗ 2
6 → ∅ ✓ 2
7 → ∅ ✗ 2
8 → ∅ ✗ 2
9 → ∅ ✗ 2
10 → ∅ ✗ 2
11 → ∅ ✗ 2
12 → ∅ ✗ 2

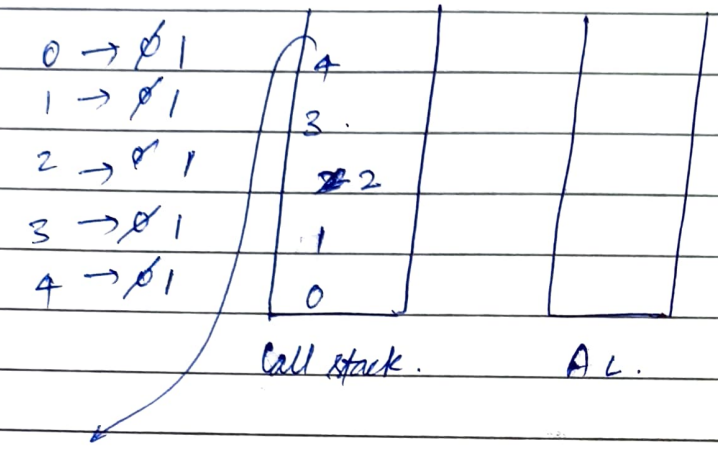


stack

APCO
Date: / /

TC-2.





now 4 call '1' and it is part of path so it is a cycle here.

Code: try to find cycle, if found return true, otherwise return false.

```
Public static boolean topoDFS(int src, int[] vis, AL<D> ans){
    vis[src] = 1;
    for(Integer e : graph[src]){
        if(vis[e] == 0){
            if(topoDFS(e, vis, ans)) return true;
        }
        else if(vis[e] == 1) return true;
    }
    ans.add(src);
    vis[src] = 2;
    return false;
}
```

APCO
Date: / /

```
public static void TopoDFS() {
    ArrayList<I> ans = new ArrayList<>();
    int[] vis = new int[N];

    boolean isCycleFound = false;
    for(int i=0; i<N; i++){
        if(vis[i] == 0){
            if(TopoDFS(i, vis, ans)){
                isCycleFound = true;
                break;
            }
        }
    }
    if(!isCycleFound) sysoln(ans);
    else sysoln("Cycle");
}
```

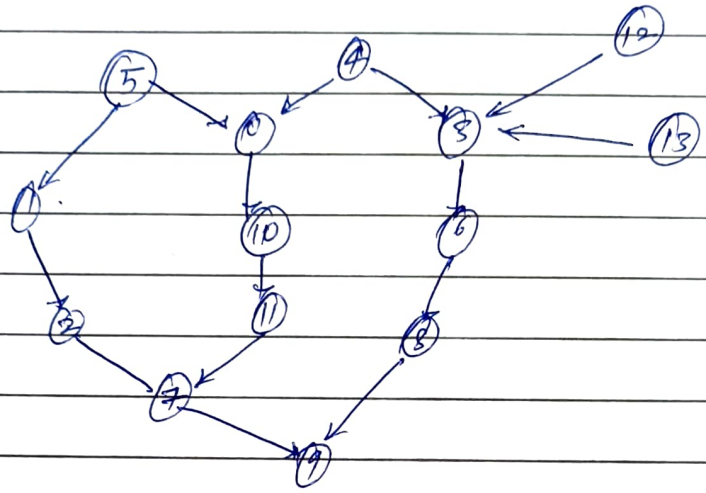
APCO
Date: / /

Using BFS (kahn's algorithm).

Very important for cycle detection in directed graph.

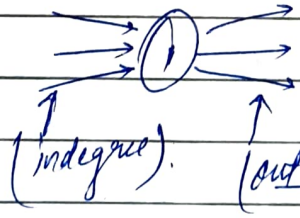
How:

TC →



Steps of kahn's alg.

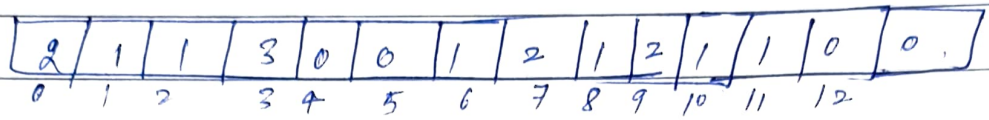
1. Find indegree.
2. enqueue the vtx having '0' indegree.
3. Resolve indegree and put the element into q if get '0' indegree else ↓ indegree by 1



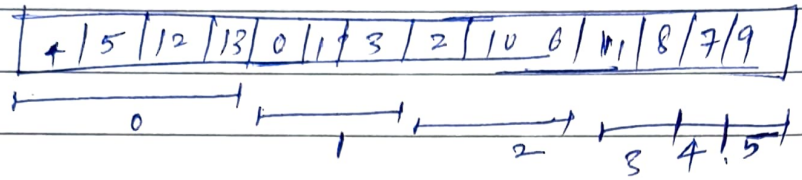
0 → 1,3
1 → 6
2 → 1
3 → 1,6,4
4 → 2,5
5 → 2,6
6 → 4

APCO
Date: / /

Indegree tells dependency.

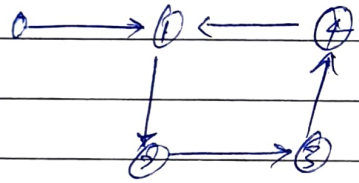


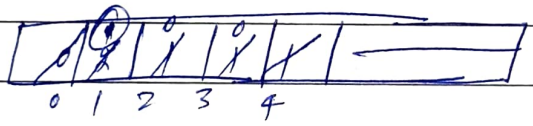
goal



① using this method can calculate cycle (detection).
② also give topological sort.

TC-2.





after removing zero we'll not be able to put other element into queue so there is cycle.

APCO
Date: / /

Code:

```
Public static void TopoBFS(){
    AL<I> ans = new AL<>();
    int[] indegree = new int[N];
    for(int i=0; i<N; i++){
        for( int e : graph[i]){
            indegree[e]++;
        }
    }
    LL<I> que = new LL<>();
    for(int i=0; i<N; i++) if(indegree[i]==0) que.addLast(i);

    while(que.size()!=0){
        int vtx = que.removeFirst();
        ans.add(vtx);

        for(int e : graph[vtx]){
            if(--indegree[e]==0) que.addLast(e);
        }
    }

    if(ans.size()!=N){ sysoln("cycle");
    else sysoln(ans);
```

Note → ~~cahn~~ kahn's algo में indegree use करते हैं बाकी normal BFS है.

APCO
Date: / /

Lecture-6 (graph)

16/10/20

Agenda:

'Kosa — directed graph BFS के qn में algo indegree use होता है)

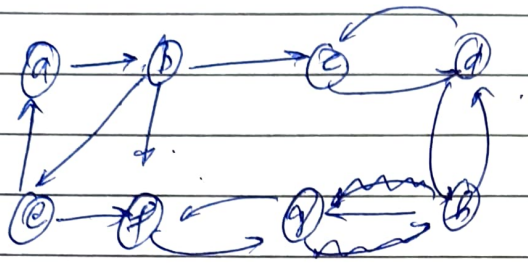
1. Strongly connected component → (good for cycle detection)
2. 329 longest increasing path in matrix → do by dp also / using BFS (kahn's algo) / using SCC.
   ↳ decreasing path.
3. ~~Union find~~ (kosaraju algorithm).
4. Bus routes. (815)
5. remain qn (694) distinct islands.

APCO
Date: / /

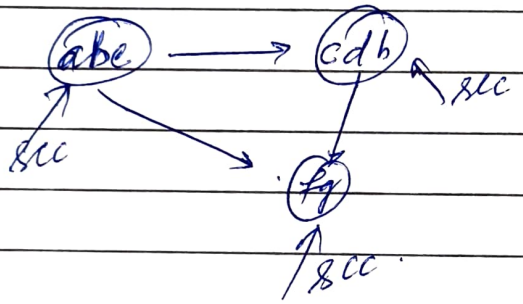
1. Strongly connected Component:

if every vto is reachable from every other vto then that is called scc.

eg.



Here scc are.



all are individual scc.

Note:- scc converts directed cyclic into directed acyclic graph

Here (abe) → ~~abe~~ bea eab, ~~bea~~ abe.
scc
all are scc

abfgf : X not scc.
bcgf g a cannot reached again.

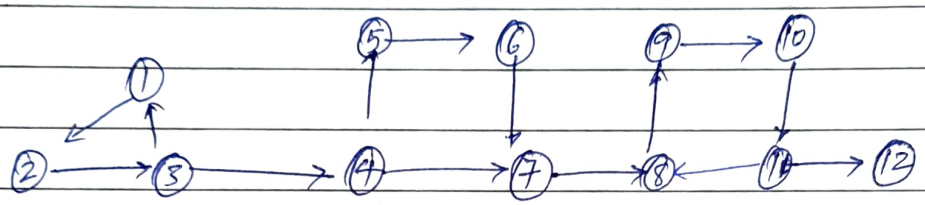
APCO
Date: / /

Kosaraju Algorithm :-

① Run topo-dfs() → ensures u come before v
② g = g' ensures storage and gives connected point (inverse the graph).
③ AL <> → dfs over topological order (not sort).

T-C-1

eg



S-1

visited

1 →
2 →
3 →
4 →
5 →
6 →
7 →
8 →
9 →
10 →
11 →
12 →

Recursion stack

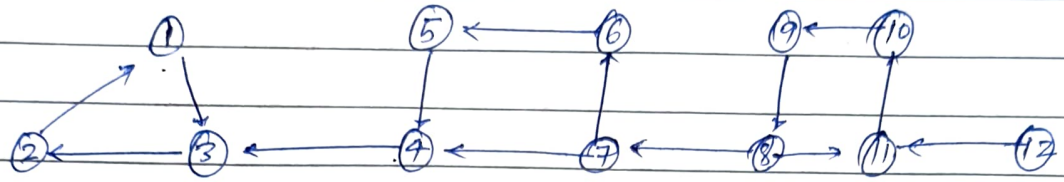
1
2
3
4
5
6
7
8
9
10
11
12

→ it is dfs topological o not sort (∵ contain cycle)

~~2~~ visit
at time of return put into stack.

APCO
Date: / /

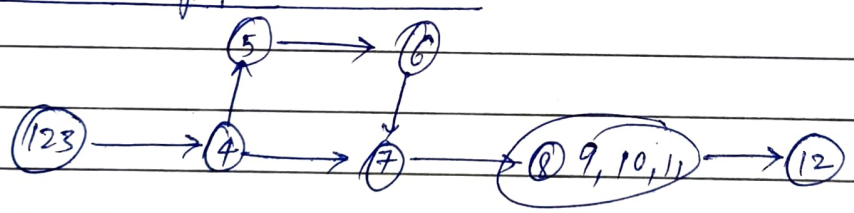
Step-2 Reverse of graph.



Step-3. Remove elements from stack and print all their connected components.

1, 2, 3.	8, 11, 10, 9.
4	
5	
6	
7	

Resultant graph after SCC.



(This is directed and acyclic graph)

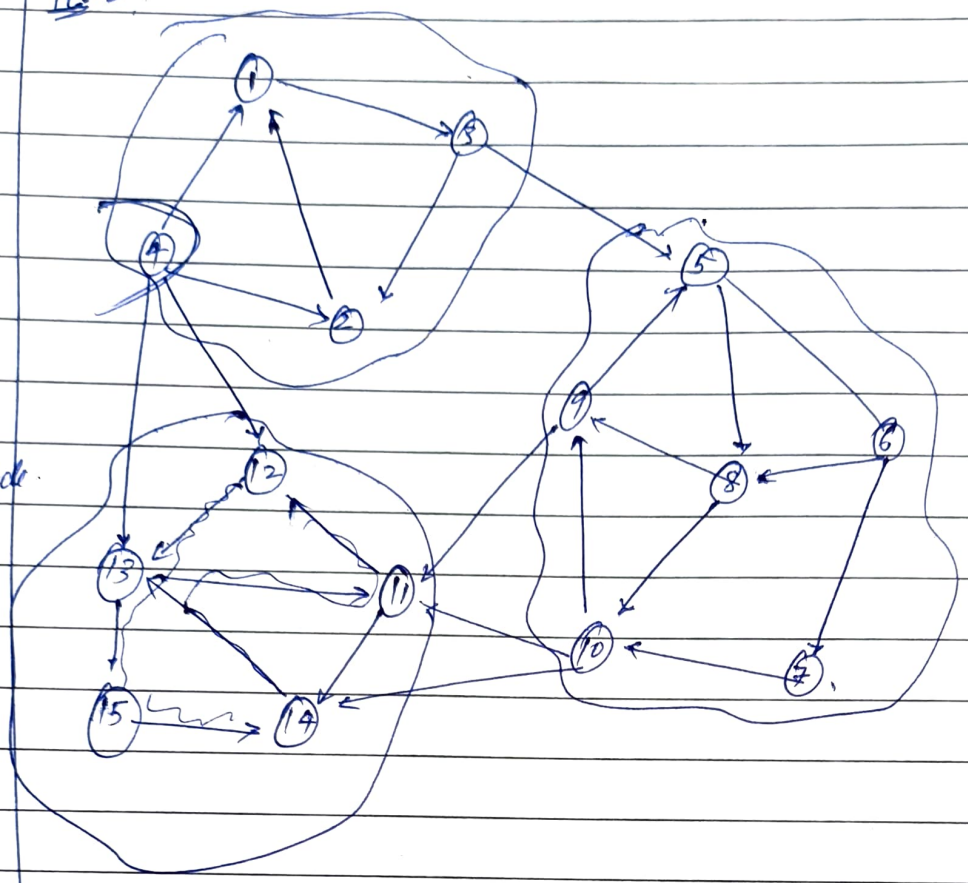
Note :- (1) It is not possible to find total no. of cycle in directed/ Undirected graph.

(2) Here we are not getting the topological sort but topological order which help to get connected comp.

(3) We cannot reverse graph without extra space.

APCO
Date: / /

Tc: 2



Cycle

→ all circled elements are connected components.

dry Run:

① Run - topo order using dfs:

4
~~3~~ 1
~~4~~ 3
~~5~~
~~6~~
~~7~~
~~8~~
10
9
11
12
14
13
15
2

← Recursive Stack

for marking visited

1	12
2	13
3	14
4	15
5	
6	
7	
8	
9	
10	
11	

APCO
Date: / /

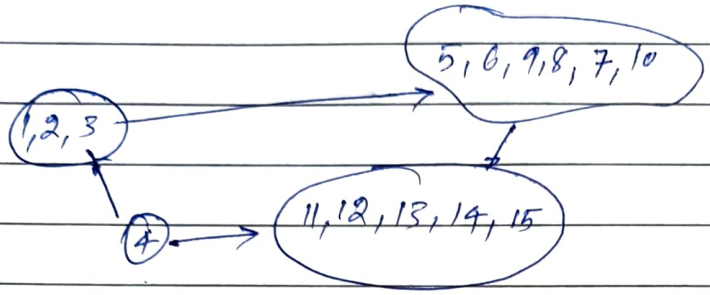
② Reverse graph and run dfs

4 components are.

1,2,3.

9, 8, 5, 6, 10, 7

11, 13, 14, 15, 12.



acyclic directed graph obtained after scc / kosaraju algorithm

// Code.

Here we are gonna write 3 fn.

① dfs_scc(make topological order of graph).

② dfs_scc2(print connected comp of each traversal)

③ Run scc with help of ① and ②

```
① Public static void DFS_SCC( int src, boolean[] vis, AL<I> path){
        vis[src] = true;
        for(int e: graph[src]){
            if(!vis[e])
                DFS_SCC(e, vis, path);
        }
        path.add(src);
    }
```

APCO
Date: / /

② Public static void dfs_scc2(int src, AL<I>[] ngraph, boolean[] vis){
vis[src] = true;
system.out.print(src + " ");

for(int e: ngraph[src]){
if(!vis[e])
DFS_scc2(e, vis, path);

}

③ kosaraju algorithm.

Public static void scc(){
// Topological order.
AL<I> path = new AL<>;
boolean[] vis = new boolean[N];
for(int i=0; i<N; i++){
if(!vis[i])
DFS_scc(i, vis, path);

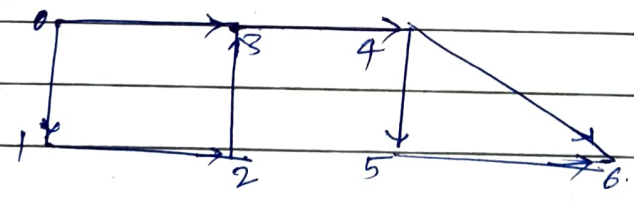
}

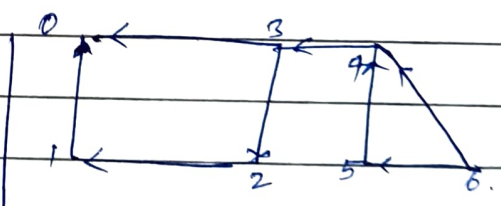
// Reverse graph.
AL<I>[] ngraph = new AL[N];
for(int i=0; i<N; i++) graph[i] = new AL<>();

APCO
Date: / /

```
for (int i=0; i<N; i++){
    for(int e: graph[i]){
        ngraph[e].add(i);
    }
}

// dfs over topological order.
vis = new boolean[N];
int count = 0;
for(int i=0; i< path.size()-1; i>=0; i++){
    if(!vis[path.get(i)]){
        count++;
        DFS_SCC2(path.get(i), ngraph, vis);
    }
}
}. // fn.
```

How graph get reverse.





0 → 3, 1
1 → 2
2 → 3
3 → 4
4 → 5, 6
5 → 6
6 → x

0 → 0
1 → 0
2 → 1
3 → 0, 2
4 → 3
5 → 4
6 → 4, 5

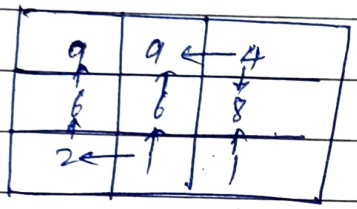
APCO
Date: / /

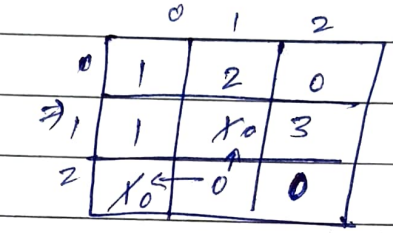
2 Que 329 // Longest ↑ing path in matrix. (using bfs, kahn's algorithm in 2-D)). Using dp.

9	9	4
6	6	8
2	1	1

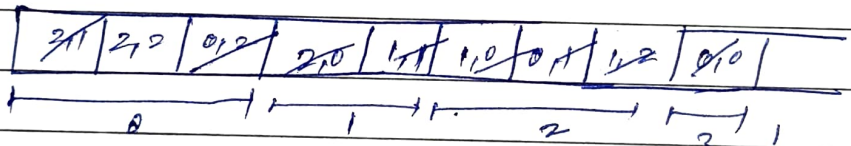
o/p = 4 for kahn's algo.

① Calculate indegree.





② Put cell having zero indegree into the que.



remove ↓to ↑ indegree of adjacent by 1 and if indegree = 0 put into que.

after emptying all element level = 4
so path length = 4.

APCO
Date: / /

// 815 Bus to destination.

```
Public int numBusesToDestination(int[][] routes, int src, int dest){
    if(src == dest) return 0;
    int n = routes.length;

    HashMap<I, AL<I>> busStandMap = new HashMap<>();
    for(int busNo = 0; busNo < n; busNo++){
        for(int stand: routes[busNo]){
            busStandMap.putIfAbsent(stand, new AL<>());
            busStandMap.get(stand).add(busNo);
        }
    }
    LL<Integer> que = new LL<>();
    HashSet<I> busStandVis = new HashSet<>();
    boolean[] busNoVis = new boolean[n];
    que.addLast(src);
    busStandVis.add(src);
    int level=0

    while(que.size() !=0){
        int size = que.size();
        while(size-->0){
            int busStand = que.removeFirst();
            for(int busNo: busStandMap.get(busStand){
                if(busNoVis[busNo]) continue;
                busNoVis[busNo] = true;
                for(int stand: routes[busNo]){
                    if(busStandVis.contains(stand)) conti
                    busStandVis.add(stand);
                    que.addLast(stand);
                    if(stand == dest) return level+1;
                }
            }
        }
        level++;
    }
    return level;
}
```

APCO
Date: / /

1 → 6.

Concept

Bus NO	Routes (Stand)
0	1, 2, 7
1	3, 6 7

① HashMap of ~~bus and~~ stand and incoming buses.

1 → 0
2 → 0
3 → 1
4 → x
5 → ^
6 → 1
7 → 0, 1

② HM of stand visited.

x	7	7	3	6

6 1 → (return (level+1) give count of buses).

boolean of bus visited.

0	

APCC

Date: /

Lecture-7 (graph) 1168, 839, 547

Agenda:

1. Union find.
2. 684 // uf on graph.
3. 1061 // find smallest lexicographic string (uf on string).
4. 200 // using union find (give insight for matrix).

APCO
Date: / /

① Union find.

(set 1 ∪ set 2) = set 3.

{2, 3, 4, 10} ∪ {1, 15, 3, 4, 10}.

if set 2 and contain then leave,

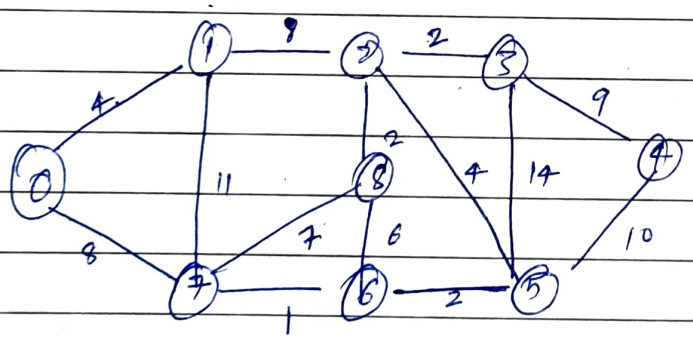
else add into set 1

set 1 = 10^3.

set 2 = 10.

set 1 ∪ set 2 = O(10). } diff complexity.

set 2 ∪ set 1 = O(10^3). }



APCO
Date: / /

Path compression.

(spanning tree) (Union find / disjoint-set) Application

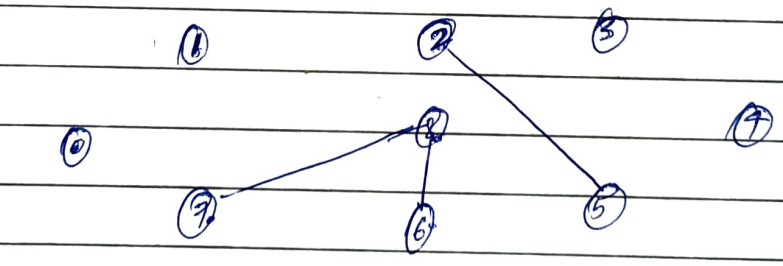
① graph → (ST.) → having no cycle.
② cycle detection.
③ ~~get~~ get connected component
④ size (component size).
⑤ MST.

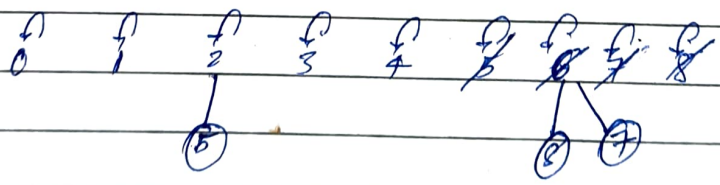
How ? union find.

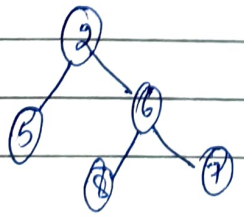
① make array of size and parent

Parent	0	1	2	3	4	~~5~~ 2	6	~~7~~ 6	~~8~~ 6
size	1	1	~~2~~	1	1	1	~~1~~ ~~2~~ 3	1	1
	0	1	2	3	4	5	6	7	8.

(u, v, w) → (0, 1, 4), (1, 2, 8), (2, 3, 7).

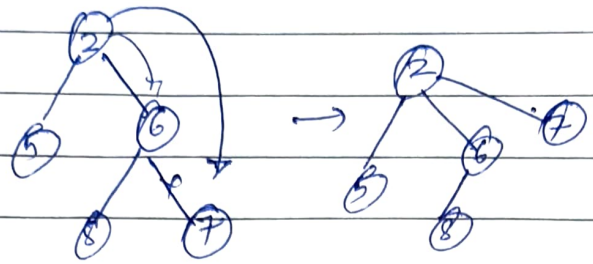






here path compression will take place.

means node get directly connect to parent (root) while returning.



Similarly two p node get connected to root directly so that union find with path compresion have complexity of

Code for compresion.

```
(1)  int find (int v) {
       if (v == parent[v]) {
         return v;
       }
       return parent[v] = find (parent[v]);
     }

     vector<int> par;
     vector<int> size;

(2)  void merge (int p1 and, int p2) {
       if (size[p1] < size[p2]) {
         par[p1] = p2;
         size[p2] += size[p1];
       }
```

APCO
Date: / /

```
else{
    par[p2] = p1;
    size[p1] += size[p2];
}
}
```

(2)
```
void unionfind(int n, vector<vector<int>> edges){
    vector<vector<Edges>> ngraph(n, vector<Edge>());

    for(int i=0; i<n; i++){
        par.push_back(i);
        size.push_back(1);
    }

    for(vector<int> a: edge){ // a = {u, v, w}.
        int gp1 = findpar(a[0]); // par of u.
        int gp2 = findpar(a[1]); // par of v.

        if(gp1 != gp2){
            merge(gp1, gp2);
            addedge(ngraph, a[0], a[1], a[2]);
        }

    }

    display(ngraph);

}
```

APCO
Date: / /

Que 684 // Find Redundant Connection.

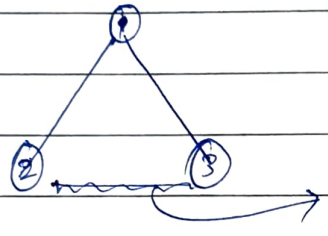
What ?

we have given edges, and have to return that edge which make cycle.

eg.

[[1,2], [1,3], [2,3]].

o/p = [2,3].



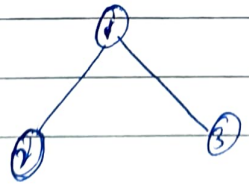
How :- ① do using union find (given only one cyclic edge)

② whenever get cycle (two node having) same parent return that edge.

③ Here merging fn (if not write doesn't matter) just speed get hamper. (anyone get merge bigger in smaller one).

dry Run = 1

par	0	1	~~2~~ 1	~~3~~ 1
size	1	1/2	1	1
	0	1	2	3



parent are same.
as they were
so there will be cycle and return that edge.

APCO
Date: / /

// 684 Code.

```
vector<int> findRedundantConnection(vector<vector<int>>& edges){
    int n = edge.size();
    for(int i=0; i<=n; i++){
        par.push_back(i);

    for(vector<int> a: edges){
        int p1 = findPar(a[0]);
        int p2 = findPar(a[1]);

        if(p1 != p2){
            par[p1] = p2;
        }else return a;
    }

    return {};

}
```

APCO
Date: / /

1061. Q1 Lexicographically Smallest String. [Locked].

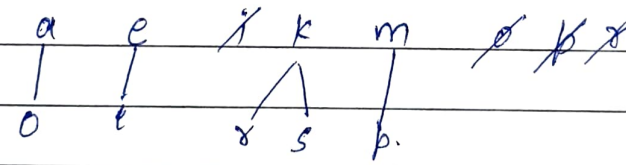
What ? A = "abc".
B = "cde".
S = "eed"
O/P = aob.

eg A = Parker, B = morris.
S = Parser.
O/P = makkek.

How :-

① Put all the character in lexicographical order.
② Club them using A and B relation.
③ Find smallest character for 'S' from A and B clubing.

Parker, morris.



Now travel on 'S' string and make answer.

APCO
Date: / /

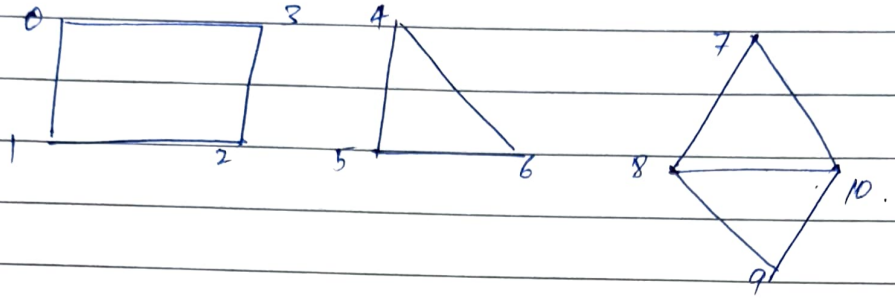
// code -

```
Public String smallestEquivalentString ( String A, B, S) {
    Par = new int [26];
    for (int i = 0; i < 26; i++)  Par[i] = i;

    for (int i = 0; i < A.length; i++) {
        int p1 = findPar ( A.charAt(i) - 'a');
        int p2 = findPar ( B.charAt(i) - 'a');

        Par [p1] = Math.min ( p1, p2);
        Par [p2] = Math.min ( p1, p2);
    }

    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < S.length; i++) {
        int p = findPar (S.charAt(i) - 'a');
        sb.append ((char) (p + 'a'));
    }

    return sb.toString();

}
```

APCO
Date: / /

Que 200 Count No of island and size.

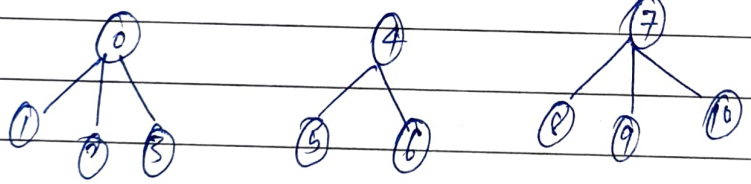
① Connected component ② No. of element in each component (UF).



0	~~0~~	~~0~~	~~0~~	4	~~4~~	~~4~~	~~7~~	7	~~7~~	~~10~~
4	X	X	1	3	1	1	4	1	1	1
0	1	2	3	4	5	6	7	8	9	10

① We'll get total comp. when i = par[i]. (gives total component present. (जितने leader उतने component).

②



② Size/area of i = par[i] will give size of each parent.

APCO
Date: / /

very Imp. Qn.

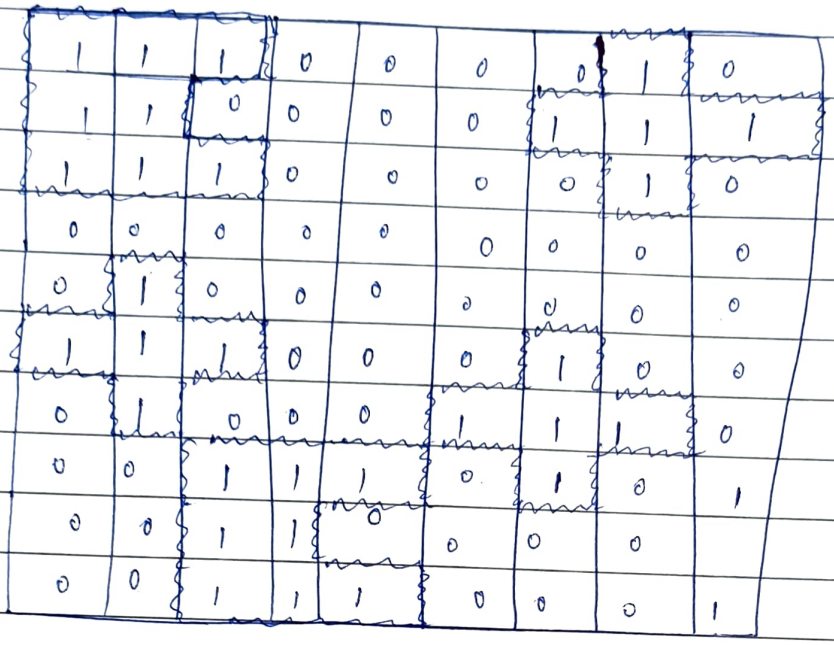
// code ① no. of island ② size of each island.

```
int numIslands(vector<vector<char>>& grid){
    int n = grid.size();
    if(n==0) return 0;
    int m = grid[0].size();
    int countIslands = 0;
    vector<vector<int>> dis = {{1,0},{0,1}};
    vector<int> size;
    for(i=0; i<n; i++){
        for(int j=0; j<m; j++){
            if(grid[i][j]=='1'){
                par.push_back(i*m+j);
                size.push_back(1);
                countIslands++;
            }else{
                par.push_back(-1);
                size.push_back(0);
            }
        }
    }

for(int i=0; i<n; i++){
    for(int j=0; j<m; j++){
        if(grid[i][j]=='1'){
            int p1 = FindPar(i*m+j);
            for(int d=0; d<2; d++){
                int x = i+dis[d][0];
                int y = j+dis[d][0];
                if(x>=0 && y>=0 && x<n, y<m &&
                    grid[x][y]=='1'){
                    int p2 = findPar(x*m+y);
                } if
}
```

APCO
Date: / /

These are not on unionfind. based on (left-2).

Qu 694 (distinct island count) [Locked].

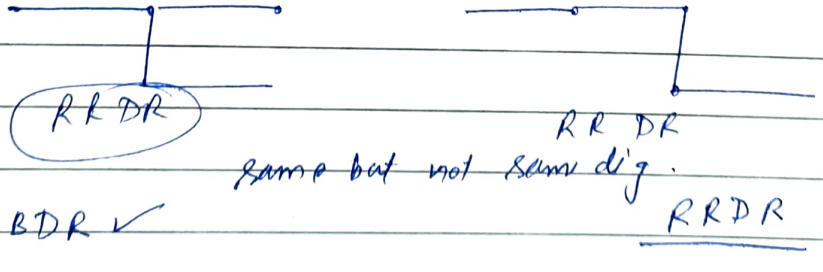


How:
1. Run recursion if recursion tree of two island same then they will be identical.
2. Here have to take 'B' for backtrack also otherwise some redundancy happen.

DDRRBUURBBBBBB. (String of recursion).
Similarly generate ~~all~~ string for all island and put it into map/set preffered

3. total element in HS will be our answer.

TC- if B don't take then what happen.



APCO
Date: / /

// Code:

```
HashSet<String> map = new HashSet<>();
int[][] dir = {{0,1},{1,0},{0,-1},{-1,0}};
char[] dirs = {'r','d','l','u'};
String shape = "";
int n=0, m=0;

① Public void dfs(int i, int j, int[][] grid){
    grid[i][j] = 0;
    for(int d=0; d<4; d++){
        int r = i + dir[d][0];
        int c = j + dir[d][1];
        if(r>=0 && c>=0 && r<n && c<m && grid[r][c]==1){
            shape += dirs[d];
            dfs(r, c, grid);
            shape += "b";
        }
    }
}

② Public int numDistinctIsland(int[][] grid){
    if(grid.length==0 || grid[0].length==0) return 0;
    n = grid.length;
    m = grid[0].length;
    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            if(grid[i][j]==1){
                dfs(grid, i, j, grid);
                map.add(shape);
                shape = "";
            }
        }
    }
    return map.size();
}
```

Based on topological sort (Sect-3).

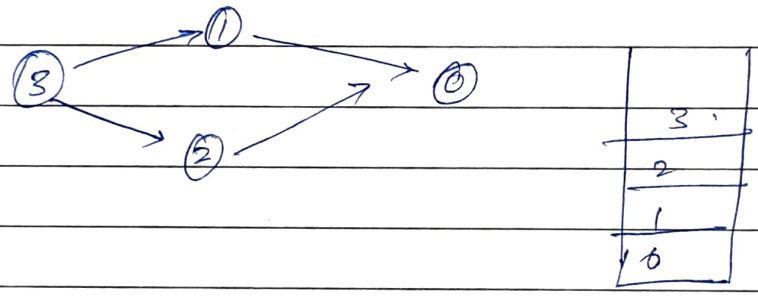
APCO
Date: / /

Que 207 / Course schedule. (Based on topological sort).

what ? given a total no. of course and they have some pre-requisite to finish them. like [0,1] have to finish 0, inorder to go to 1.

eg. num Course = 4.
Pre-requisite [[1,0], [2,0], [3,1], [3,2]];

Note :- follow Any one standard either (u, v) or (v, u) and change your answer if you are taking opposite standard.



How :- ① Calculate indegree.
② run BFS.

2	0	0	0
0	1	2	3

Put zero indegree element into que

bfs que

3	1	2	0

answer que.

3	1	2	0

count = courses. so we will be able to finish course.

APCO
Date: / /

// Code -

```
Public boolean canFinish(int N, int[][] arr){
    AL<I>[] graph = new AL[N];
    for(int i=0; i<N; i++) graph[i] = new AL<>();

    int[] indegree = new int[N];
    for(int[] a: arr){
        indegree[a[1]]++;
        graph[a[0]].add(a[1]); // making graph
    }
    LL<I> que = new LL<>();
    for(int i=0; i<N; i++) if(indegree[i]==0)
        que.addLast(i);

    int count=0;
    while(que.size()!=0){
        int vtx = que.removeFirst();
        count++;
        for(int e: graph[vtx]){
            if(--indegree[e]==0) que.addLast(e);
        }
    }
    return count==N;
}
```

APCO
Date: / /

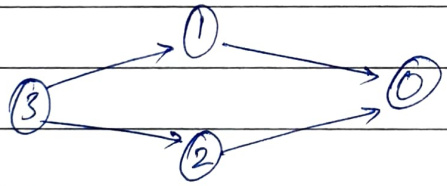
Qu 210 (Schedule - II)

What? given 'n' course and pre-requisite (0 to n-1)
return ordering of subject in which you finish courses.

How: whenever we have anything like ordering / pre-requisite then simply use topological.
① dfs.
② Bfs (khan's algorithm).

Here i'm using Bfs (khan's algorithm).

eg. N = 4.
pre-requisite = [[1,0],[2,0],[3,1],[3,2]];.



① indegree.

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

② put into que. of '0' indegree.

~~3~~	1	2	0

ans. return ans.

0	1	2	3

APCO
Date: / /

M code:

```
Public int [] findOrder(int N, int [][] arr){
    AL<I> graph[] = new AL [N];
    for(int i=0; i<N; i++) { graph[i] = new AL<>(); }

    int [] indegree = new int [N];
    for(int [] a : arr){
        indegree [a[1]]++;
        graph[a[0]].addLast(a[1]);
    }

    LL<I> que = new LL<>();
    for(int i=0; i<N; i++) if(indegree[i]==0)
        que.addLast(i);

    int [] ans = new int[N];
    int idx = N-1;
    while( que.size() !=0){
        int vtx = que.removeFirst();
        ans[idx--] = vtx;

        for(int e : graph [vtx]){
            if(--indegree[e]==0) que.addLast(e);
        }
    }

    if(idx==-1) return ans;
    return new int[0];
}
```

APCO
Date: / /

Lecture-8 (graph) (19-08-2020)

Agenda:

1. 305 // Count Island-2
2. 839 // Similar string count
3. 990
4. 1135
5. ~~1168~~ kruskal algorithm.
6. 1168 // water supply cost
7. Mr. president on Rachel Earth.

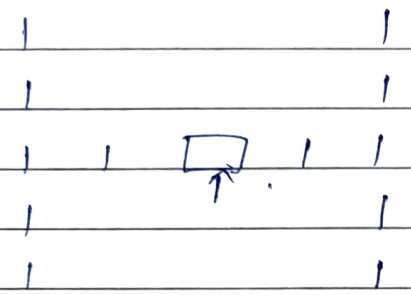
~~Oct, Nov~~ Oct
Nov
dec
Jan
Feb
Mar

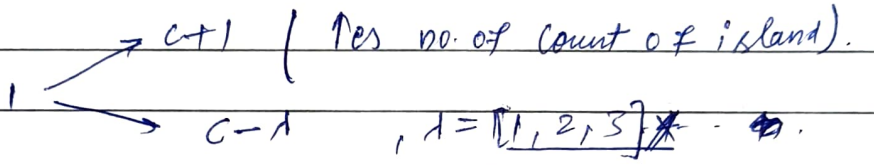
Apr
May
June
July
Aug → AR/ML

APCO
Date: / /

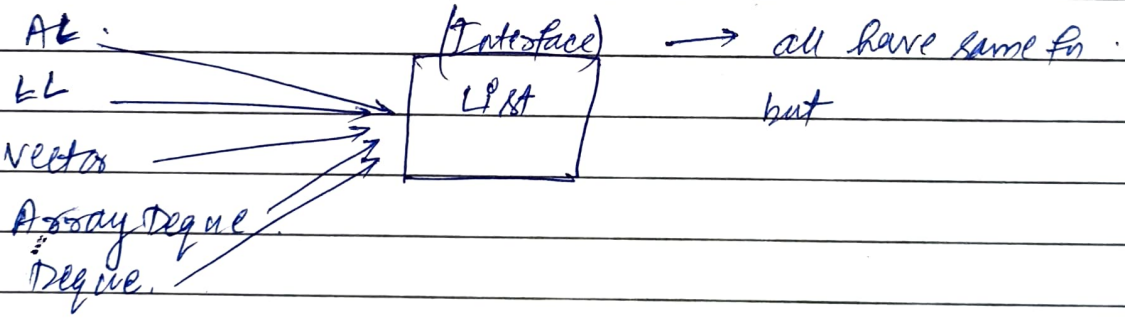
8

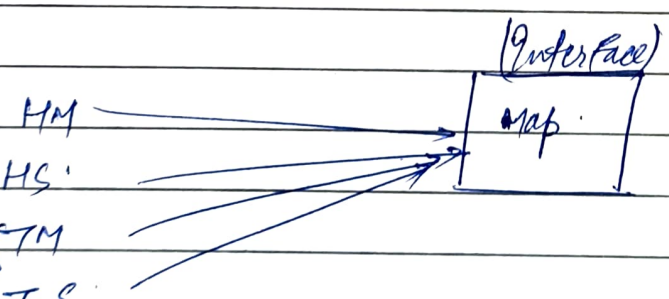
Qw 305 (using union find).





stl in java-





sorted data हेतु TS (BST).

108, 130, 200, 399, 547,
721, 959, 1202, 1319

APCO
Date: / /

Q-2 dfs

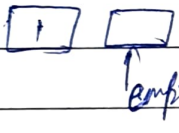
I place ~~this~~ call gcc. and it will give total island count.

$(n \times m) \times k$.

$tf = k \times O(4)$. ⟶ due to path comp if (node $< 10^{600}$).
worst case = $k \log n$. if (node $> 10^{600}$).

Here 2 cases arise.

TC1



APCO
Date: / /

3 No. of islands.

what?

A grid of 'm' row, 'n' column filled with water. we have to perform add land operation which turn water at position row, col into a land. given a position to operate count the number of island after each add land operation.
An island is surrounded by water and formed by connecting adjacent land horizontally/vertically.

I/P: n = 3, m = 3,
position = [[0,0], [0,1], [1,2], [2,1]];

O/P = [1, 1, 2, 3]

Code

```
Public List<Integer> numIsland2 (int n, int m, int[][] Positions) {
    par = new int[n*m];
    for(int i=0; i< n*m; i++) { par[i] = i;

    int islandCount=0;
    int[][] dir = {{0,1},{0,-1},{1,0},{-1,0}};

    int[][] mat = new int[n][m];
    List<I> ans = new AL<>();
```

APCO
Date: / /

```
for(int[] a: positions){
    int i = a[0];
    int j = a[1];
    if(mat[i][j]==0){
        int p1 = findPar(i*m+j);
        mat[i][j] = 1;
        islandCount++;

        for(int d=0; d<4; d++){
            int r = i + dir[d][0];
            int c = j + dir[d][1];
            if(r>=0 && c>=0 && r<n && c<m
                && mat[r][c]==1){
                int p2 = findPar(r*m+c);
                if(p1!=p2){
                    islandCount--;
                    par[p2] = p1;
                }
            }
        }
    }
    ans.add(islandCount);
}
return ans;
}
```

→ ~~छो~~ grp आ जाए तो gp for union find.
gives best complexity.

May
June
July
Aug

APCO
Date: / /

Qn ~~305~~ 839.

What :- Two string 'x' and y are similar if we can swap two letter of 'x' so that it equals y.
Also two string 'x' and 'y' are similar if they are equal.

eg. tars → rats similar ✓
rats → arts ✓
tars → star ✗ similar.

I/P : ["tars", "rats", "arts", "star"].
O/P : 2.

How :-
Here we will check whether string are equal, if equal then make their parents equal and ↓ment count by 1.

eg 1:-

tars → rats, arts, star
rats → arts, star
arts → star
} (all possible combination)

tars → rats → arts.

tars (parent: tars) rats (parent: rats) arts (parent: arts) star (parent: star)

tars — rats, arts ; star } so 2 grp present = O/P = 2 Ans.

APCO
Date: / /

// Code.

① Public boolean isSimilar(String a, String b) {
int count = 0;
for(int i = 0; i < a.length; i++) {
if(a.charAt(i) != b.charAt(i)) && ++count > 2) {
return false;
}
}
return true;
}

② Public int numSimilarGroups(String[] A) {
int n = A.length();
Par = new int[n];
for(int i = 0; i < n; i++) Par[i] = i;
int count = n;
for(int i = 0; i < n; i++) {
for(int j = i+1; j < n; j++) {
if(isSimilar(A[i], A[j])) {
int P1 = findPar(i);
int P2 = findPar(j);
if(P1 != P2) {
Par[P1] = P2;
count--;
}
}
}
}
return count;
}

APCO
Date: / /

Spanning tree :- Spanning tree of graph is subgraph that is tree and connects all the vertices together.

→ A single graph can have many spanning tree.

→ Mst has (V-1) edges (V = Vertices of graph).

Application of Mst are.

① Network design.
② Approximation algorithm for NP-hard problems.
③ Indirect application.
- → max bottleneck path.
- → LDPC code for error correction
- → Image registration with Renyi entropy.
- → Real time face verification.

④ Cluster analysis.

Prim, dijkstra → BFS.

APCO
Date: / /

Kruskal algorithm: (is just union find). eg 1168, MR president on Hackerearth.

min (ST1, ST2, ST3 . . . . .) ← spanning tree.

$MST = \min\left(\sum_{i=0}^{V-1} G[i]\right)$.

graph is edges of min sum such that no cycle created is called mst.

→ kruskal is just part node in sorted order on weight.

// Code:

```
public static void KruskalAlgo(int No. of vertex, int[][] edges){
    graph.resize(No of vertex);
    for(int i=0; i<= No of vertex; i++){
        graph[i] = new AL<>();

    Arrays.sort(edges, (a,b) -> {
        return a[2] - b[2];
    });

    unionFind(No. of vertex, edges); // same as previous Union Find.
    display(graph);
}
```

APCO
Date: / /

Greedy Algorithm :-

It is a method that follows problem solving technique of making the locally optimal choice at each stage with the hope of finding a global optimum.

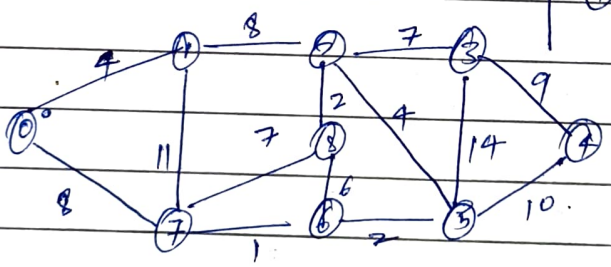
App of Krushal.

1. Landing Cables.
2. Tv Network.
3. Tour operation.
4. LAN Network.
5. A Network of pipe for gas/water.
6. electric grid.
7. Single-link cluster.

Prism.

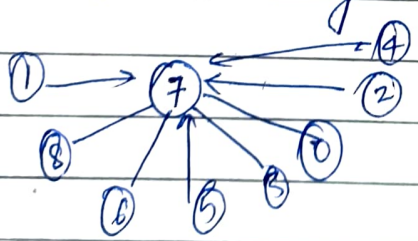
1. All places where krushal used / also in denser graph of krushal
2. N/w for road/Rail track connecting cities.
3. Irrigation channel / mw towers
4. designing fibre optic grid/ICs
5. Travelling salesman.
6. cluster Analysis.
7. Path finding algo used in (AI)
8. game development
9. Cognitive science.

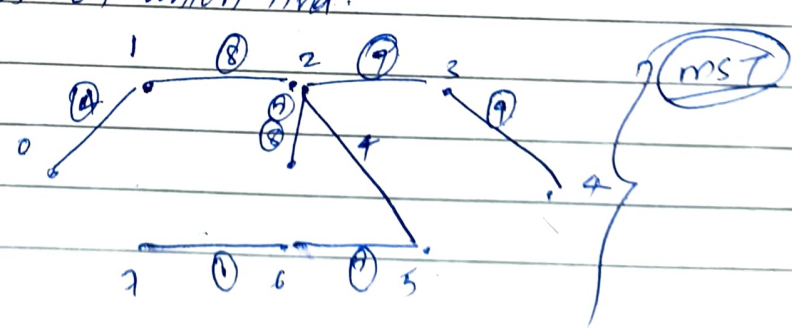
given



Sort edges on wt. basis. | 1 | 2 |

Note:- o/p graph doesn't get created on real just connection are made in array as of union find.





APCO
Date: / /

Que 1168. [water supply]. [locked]

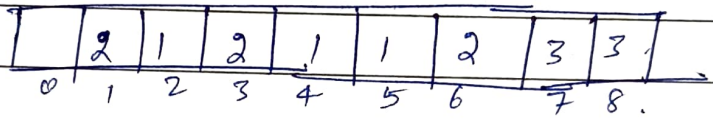
What There is 'n' houses in village, have to supply water in all house by building well / laying pipe in minimum cost.

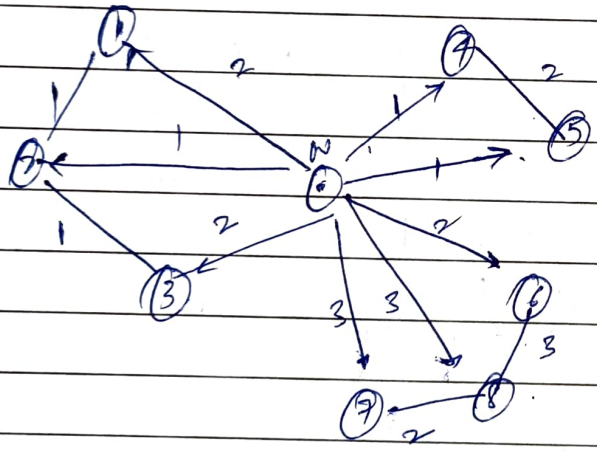
I/P:- n = 3, well = [1, 2, 2]. // well बनाने का cost

pipe = [[1, 2, 1], [2, 3, 1]].
[u, v, c]

u → v का cost

How:-





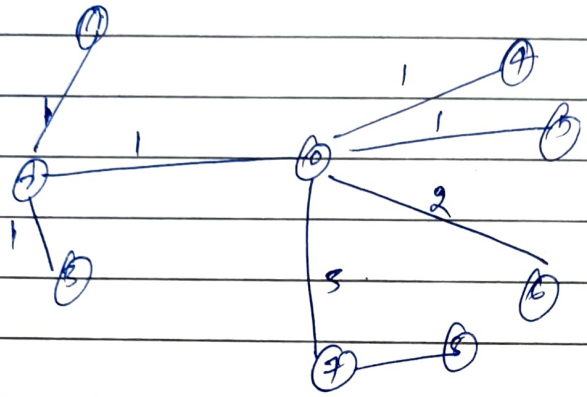
→ Instead of making well at vtx we represent well का cost in terms of edge.

→ represent cost of well in terms of edge.

APCO
Date: / /

Algo - ① Make edges of well to vto and put them and actual pipe in arraylist
② Sort them.
③ Run kruskal. get mst.

Eg:



// Code -

```
Public int minCostToSupplyWater(int n, int[] wells, int[][] pipes) {
    AL<int[]> edges = new AL<>();
    for(int i=0; i<n; i++){
        edges.add(new int[]{0, i+1, wells[i]});
    }
    for(int[] e: pipes) edges.add(e);
    Collections.sort(edges, (a,b) -> {
        return a[2]-b[2];
    });
    par= new int[n+1];
```

APCO
Date: / /

```
// Arrays.fill(par, -1);
for(int i=0; i<n+1; i++) par[i]=i;

int cost = 0;
for(int[] e : edges){
    int p1 = findPar(e[0]);
    int p2 = findPar(e[1]);

    if(p1 != p2){
        par[p1] = p2;
        cost += e[2];
    }
}
return cost;
}
```

APC
Date: /

Lecture-9. (21-10-2020)

Agenda:

1. Prims algorithm
   - O(n²) implementation.
   - standard implementation.
2. dijekstra
   - implementation standard.
3. Bellman Ford
   - ① O(n²) implementation (extra space of array).
   - ② standard implementation.
4. On on dijekstra and Prims.
   - ① 787
   - ② 743.

APCO
Date: / /

(dijekstra, bellmanford similar).

Prims. (very important algo).

why :- ① graph बना कर देता है (mst)
② Run on BFS.
③ put edges in pq. and remove from pq.
④ put vtx into que if not visited.
⑤ complete graph का wt min होता है while in dijekstra हर एक node min cost से visit होता है.

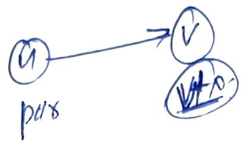
① Our implementation.

It have worst complexity but very easy to implement.

→ just apply BFS and keep pq instead of pq वाला

Algo. ① queue que.
② que ← (u, v, w).
③ while(que.size() != 0)
③.① rem vtx ← que. edge.
③.② if cycle then continue.
③.③ marks vtx.
3.3.1 for all unvisited nbr add nbr in que.

// code Public class PrimsPair {
int par = 0;
int vtx = 0;
int w = 0;
primsPair(int vtx, int par, int w) {
this.par = par;
this.vtx = vtx;
this.w = w;
}
}



मैं जहाँ जा रहा हूँ
that is vtx

APCO Date: / /

Eg:

```
Public static void primsAlgo(AL<E>[] graph, int N){
    AL<E>[] ngraph = new AL<>[N];
    for(int i=0; i<N; i++) ngraph[i] = new AL<>();

    boolean[] vis = new boolean[N];
    PriorityQueue<primspair> que = new PQ<>((a,b) -> {
                return a.w - b.w; // this-other
                                   default.
    });
    que.add(new primspair(0,-1,0));
    int EdgeCount = 0;
    while(EdgeCount <= N-1){
        primspair pair = que.remove(); } O(log E).

        if(vis[pair.vtx]) continue;

        if(pair.par != -1){
            addEdge(ngraph, pair.par, pair.vtx, pair.w);
            EdgeCount++;
        }
        vis[pair.vtx] = true;
        for(Edge e: graph[pair.vtx]){
            if(!vis[e.v]){
                que.add(new primspair(e.v, e.pair,
                                      e.w));
            }
        }
    }
    display(ngraph, N); // for display fn
}
```

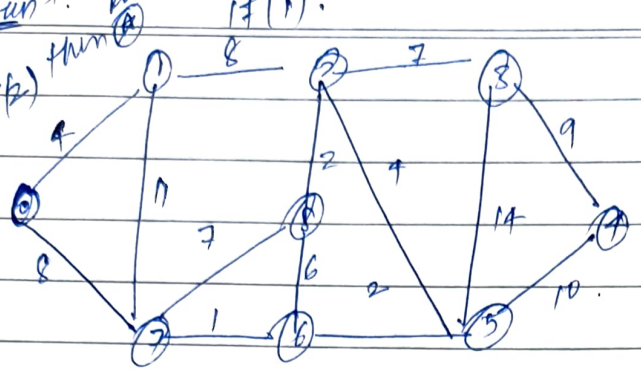
log E + E log E — add to pq (E edges) का

$O(V(\log E + E\log E))$ ← complexity

APCO
Date: / /

dry Run:- Removed from if(1).

if(2) then



pq = | 0,7,0 | 1,0,4 | 7,0,8 | 2,1,0 | 7,1,11 | 8,7,7 | 6,7,1 |

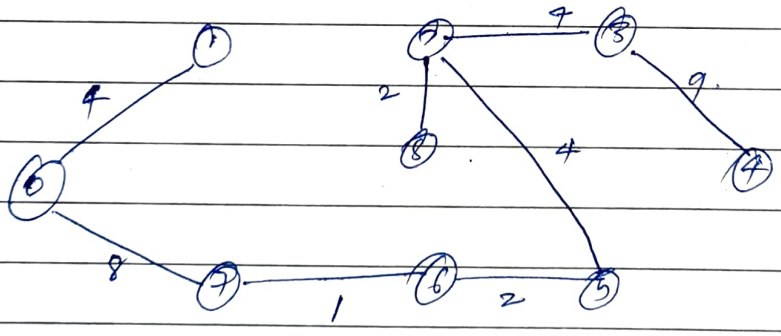
| 8,6,6 | 5,6,2 | 4,5,10 | 8,5,14 | 4,5,4 | 3,2,9 |

| 8,2,2 | 3,2,7 | 4,3,9 |

visited | T | T | T | | | T | T | T | - |

0 1 2 3 4 5 6 7 8.

→ keep track of edges which added to graph don't add again.



Here mst and (2) = 8 + 1 + 2 + 4 = 15 } so both
while min cost to (2) = 12
Prims and dijkstra are different.

→ Note :- Start from any node we'll get same (mst).

BFS → ① Prims] → Mainly find MST.
Union find → ② Kruskal]

APCO
Date: / /

② Standard Implementation of prims.

// Code.

```
Public static void prims_algo (ArrayList<Edge>[] graph, int N){
    ArrayList<Edge>[] ngraph = new ArrayList[N];
    for(int i=0; i<N; i++) ngraph[i] = new ArrayList<>();

    boolean[] vis = new boolean[N];
    int dis[] = new int[N]; // not need but
    Arrays.fill(dis, (int)1e5);

    PQ <primsPair> que = new PQ(){(a,b)->{
        return a.w - b.w;
    });

    que.add(new primsPair(0,-1,0));
    int EdgeCount=1;
    while(EdgeCount<=N-1){
        primsPair pair = que.remove();
        if(vis[pair.vtx]) continue;

        if(pair.par != -1){
            addEdge(ngraph, pair.par, pair.vtx,
                    pair.w);
        }

        vis[pair.vtx] = true;
```

APCO
Date: / /

```
for (Edge e : graph[pair.vtx]) {
    if (!vis[e.v] && e.w < dis[e.v]) {
        dis[e.v] = e.w;
        que.add(new primspair(e.v,
            pair.vtx, e.w));
    }
}
} // while loop

display(ngraph, N);
}
```

Prims algo :- It says pq. में सबसे at max vtx होने चाहिए node हो if

→ dist array takes care of wt. and put only smaller wt for that vtx into the pq.
→ Here if if put dist array then boolean की नहीं रखते तो काम चल जाएगा

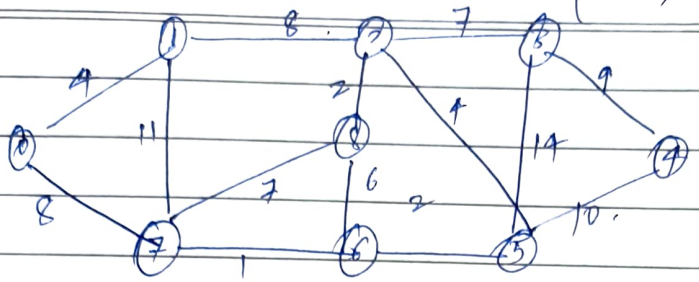
Remove (head / Root) → logn.

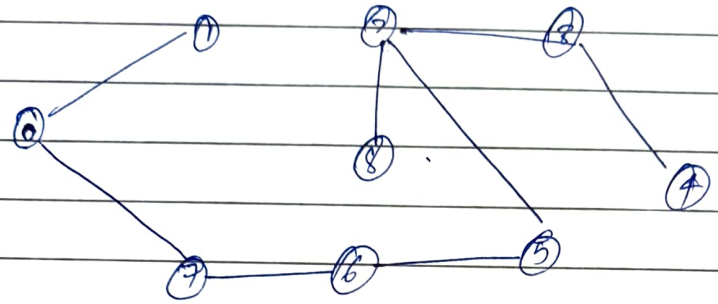
Remove (Particular object) = n logn

(find) (remove)

APCO

Date: / /





node	dist
0	0
1	∞ 4
2	∞ 8.
3	∞
4	∞
5	∞
6	∞ 1
7	∞ 8.
8.	∞. 76

boolean:

0, 1, 0	1, 0, 4	~~7, 0, 8~~	2, 1, 8	~~8, 7, 7~~	6, 7, 1	

8, 6, 6

① make redundancy → to check this. have to keep boolean.

or to update pq. to remove already

use avl tree /

→ greater element in pq.

APCO
Date: / /

Complexity Analysis:-

① Standard prims.

primsPair pair = que.remove(); → log v —①

→ Here we have assumed vto having higher wt. are updated by lower wt. so in que only 'v' vto are present in worst case.

```
for( Edge e : graph[pair.vto] {                    —②
    if(!vis[e.v] && e.w < dis[e.v]) {
        dis[e.v] = e.w;
        que.add(new primsPair(e.v, pair.vto, e.w));
    }
}
```

→ E edge are checked 2ice.

→ E log v or for insertion into pq.

+ E

total E edges.

for backing edge which are already checked.

so combined complexity = O(E log v + v log v)

$$O(v \log v + E \log v) \quad —③$$

APCO
Date: / /

② our prims / Easy version for numericals.

```
while( que.size() != 0 ){
    primsPair pair = que.remove();   (E log E) — ①
```

because there are 'E' edges in the que and removing cost log E for E edges.

```
for( Edge e : graph[ pair.vto ] ){
    if( !vis[ pair.vto ] ){
        que.add( new primsPair( e.v, pair.vto,
                                e.w ) );
    }
}
```

$O(E \log E + E)$ → ①

E edges to add into pq. for each vto again visiting but doesn't get put into que, so only 'E' times for loop

c-1 → if our while loop run till que.size.

$O(E \log E + E \log E) \Rightarrow O(2E \log E)$.

c-2 if while loop run till ~~que.size~~ vto times.

$O(V \log E + E \log E)$

que size is of Edge

loop runs 'V' times.

APCO
Date: / /

→ In Worst case ① prims ~~and this~~ standard ② our prims ~~③ both all behave same~~.

In densed graph.

when one vtx is connected to every other vtx.

$$E = \frac{V(V-1)}{2} = \frac{V^2 - V}{2}$$

$$\boxed{E = V^2}$$

our prims.

C-1 loop = given size.

$O(E \log E + E \log E)$.

$2(2V^2 \log V)$

$= 4V^2 \log V$.

$\boxed{O(V^2 \log V)}$ ④

C-2. loop = V to time

$O(V \log E + E \log E$

$2V \log V + 2V^2 \log V$

$\boxed{= O(V^2 \log V)}$ ④.

Standard.

$(V+E) \log V$

$O(V \log V + V^2 \log V)$.

$\boxed{O(V^2 \log V)}$ ④.

$\boxed{\text{In densed graph both prims behave same.}}$ ④.

APCO
Date: / /

Kruskal: Complexity

$E \log E + E \log V$.

$E \log E$ → Sorting of E edges.

$E \log V$ → For E edges.

→ parent find करना

Worst case:

$$O(V^2(\log V^2 + \log V))$$

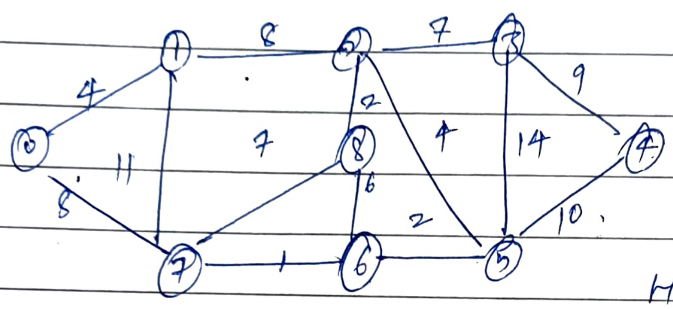
$$O(3V^2(\log V)) \quad (*)$$

Prims	Kruskal
(1) Complexity $O((V+E)\log V)$.	(1) Complexity $O(E(\log E + \log V))$.
(2) In densed graph. ($E = V^2$). $O(V \log V + V^2 \log V)$ (*) $O(V^2 \log V)$	(2). In worst case. $O(3V^2(\log V))$ (*)
(3) Used in densed graph.	(3) Sparse graph.
(4) It works always better than kruskal.	(4) It is very easy to implemen than prism so, use when sparse graph. (5) In sparse both behave same

dijekstra :-

dijekstra is same as prism

1. pq. is sorted on the basis of wt. so far wsf.
2. don't need to make graph here.
3. Here don't need visited array
4. dist array is must here.

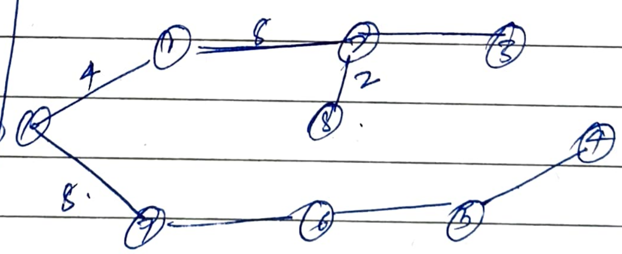


Here have to keep (u, v, wsf).

	wsf.	vis	par
0	0		-1
1	~~∅~~ 4		0
2	~~∅~~ 12		1
3	~~∅~~ ~~25~~ 19		7
4	~~∅~~ 21		5
5	~~∅~~ 11		6
6	~~∅~~ 9		7
7	~~∅~~ 8.		0
8.	~~∅. 15~~ 14		7

0, -1, 0	~~1,4~~	~~7,8~~	8,15	~~6,9~~	~~2,12~~	~~5,11~~

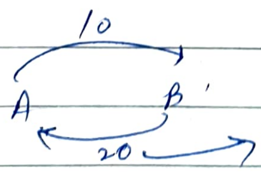
3,25	4,21	3,19	~~8,14~~



→ here dist array tells '0' के respect से any v to पर पहुँचने का cost

→ In prims no need of reference.

→ visited रखना standard है (वैसे चल जाएगा बिना visited के भी)



दोबारा visit करना पड़ेगा costly होगा

APCO
Date: / /

→ Here dist array gives किसी भी vtx तक reach करने का min cost w.r.t. any reference vtx (eg. 0).

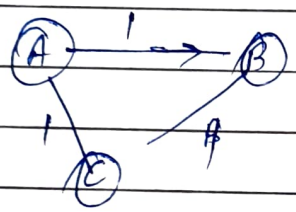
// Code:

dijekstra graph can be mst but mst need not to be dijekstra.

→ dijekstra / Bellman ford run in both directed / undirected.

→ cycle तिका visited के भी बच जाती है.

→ Undirected

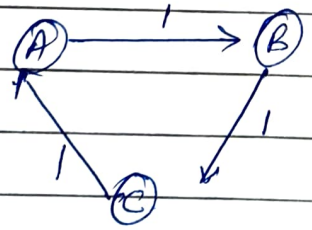


	Cost
A →	0
B →	1
C →	1

~~(A,0)~~ ~~(B,1)~~, (C,1)

↑ can't put B and C with wt (2).

→ directed.



A → 0
B → ~~∞~~ 1
C → ~~∞~~ 2

(A,0), (B,1), (C,2).

APCO
Date: / /

Code

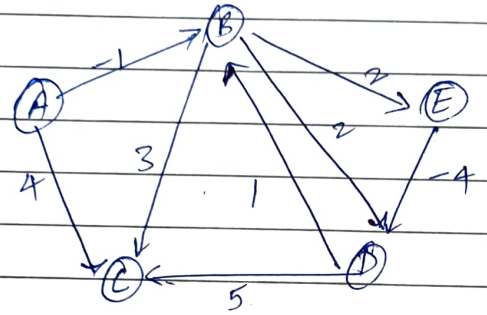
```
Public static void dijekstraAlgo(ArrayList<Edge>[] graph, int N){
    boolean[] vis = new boolean[N];
    int dis[] = new int[N];
    int[] par = new int[N];

    Arrays.fill(dist, (int)1e8);
    Arrays.fill(par, -1);

    PriorityQueue<dijekstraPair> que = new PQ<>((a,b)->{
                return a.wsf - b.wsf; // this->other
            });

    que.add(new dijekstraPair(0,-1,0));
    dis[0] = 0;
    int EdgesCount = 1;

    while(EdgesCount <= N-1){
        dijekstraPair pair = que.remove();
        if(vis[pair.vtx]) continue;

        if(pair.par != -1) EdgesCount++;

        vis[pair.vtx] = true;
        par[pair.vtx] = pair.par;

        for(Edge e : graph[pair.vtx]){
            if(!vis[e.v] && pair.wsf + e.w < dis[e.v]){
                dis[e.v] = pair.wsf + e.w;
                que.add(new dijekstraPair(
                    e.v, pair.vtx, pair.wsf
                    e.w));
            } // if
        } // for
    } // while
} // fn
```

APCO
Date: / /

Bellman Ford Algo.

It is for graph having -ve edge.
(Also detect -ve cycle). in directed graph.

① Our Implementation. (Ford On purpose).



→ It is a dp problem.
→ कहीं भी पाथ में max (v-1) edges.
→

AB = -1
AC = 4
BE = 2
BD = 2
BC = 3
DB = 1
ED = -4
DC = 5

	0	1	2	3	4	i get update in vth iteration
	od	od	nd			
A	0	0	0	0	0	
B	∞	-1	-1	-1	~~-2~~	
C	∞	4	~~2~~	~~2~~	~~2~~	
D	∞	∞	~~1~~	1	~~-3~~	
E	∞	∞	~~1~~	1	1	

AB = 0 + 0(-1) = -1
AC = 0 + 4 = 4

mean there is -ve cycle

किसी भी edge पर पाथ ≤ (k-1) edge + current edge = distance.

APCO
Date: / /

// Code - // edges [u, v, w] - - - }.

```
public static void BellmanFordAlgo(int[][] edges, int N, int src){

    int[] dis = new int[N];
    Arrays.fill(dist, (int)1e8);
    dist[src] = 0;
    boolean negativeCycle = false;

    for(int i=0; i<=N; i++){
        boolean isAnyUpdate = false;
        int[] ndis = new int[N];
        for(int j=0; j<N; j++) ndis[j] = dis[j];

        for(int[] e : edges){
            if(dis[e[0]] != (int)1e8 &&
               dis[e[0]] + e[2] < ndis[e[1]]){
                ndis[e[1]] = dis[e[0]] + e[2];
                isAnyUpdate = true;
            }
        }
        dis = ndis;

        if(isAnyUpdate && i==N) negativeCycle = true;
        if(!isAnyUpdate && i<N) break;
    }
}
```

APCO
Date: / /

② Standard Bellman Ford.