Doubt Class With Lakshay Bhaiya [Graph]

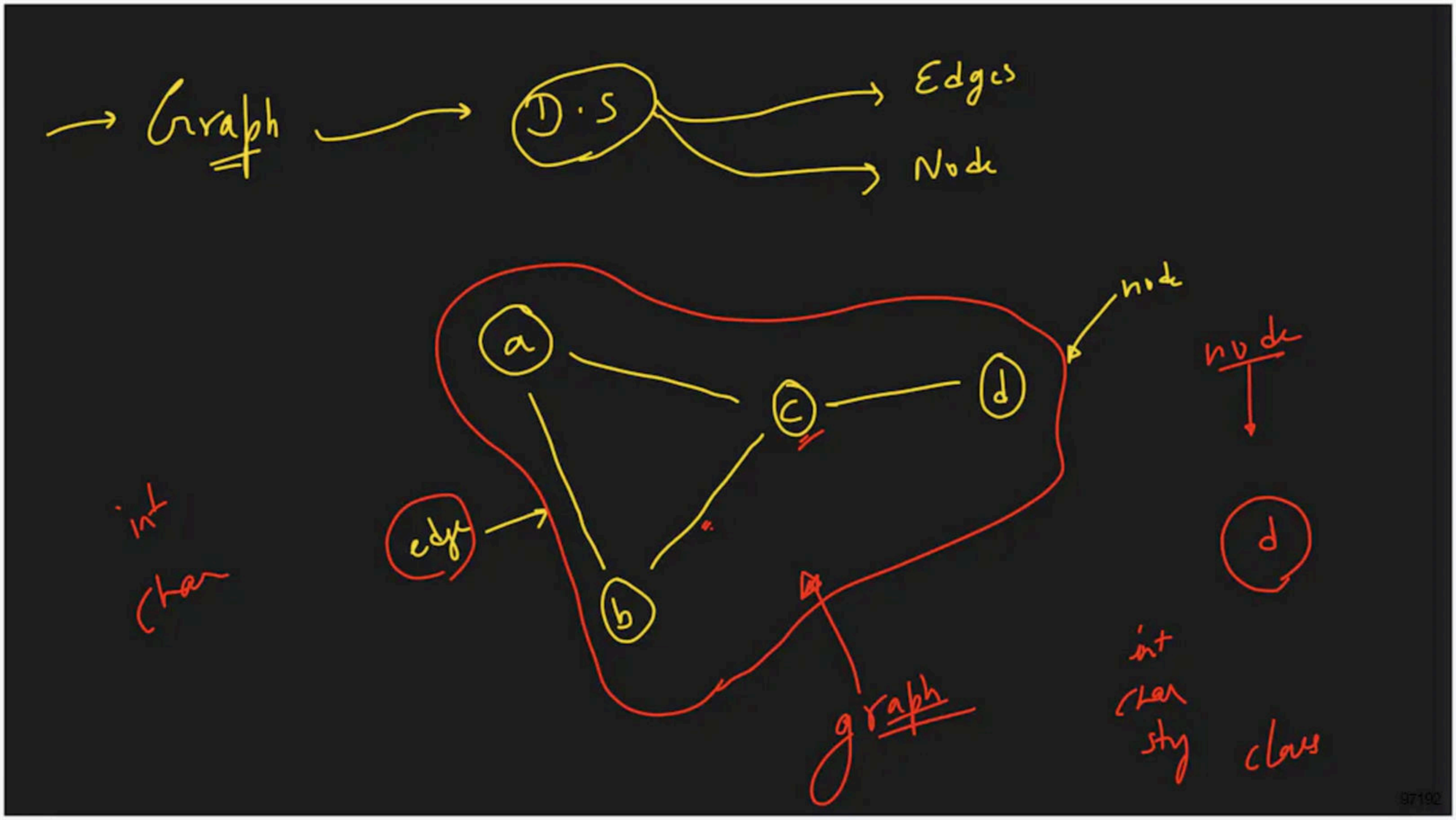
Special class

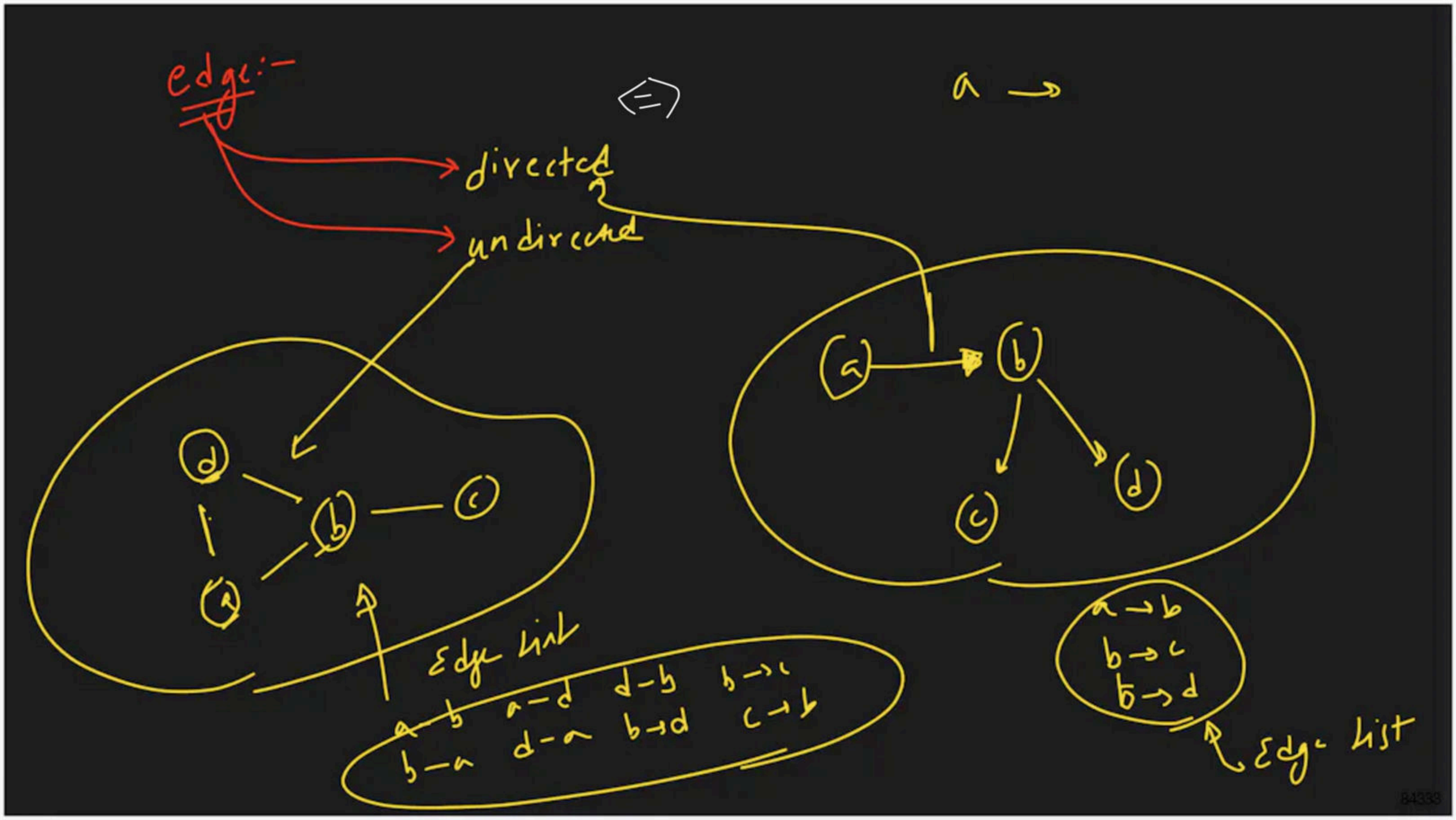
Love Babbar • Feb 11, 2024

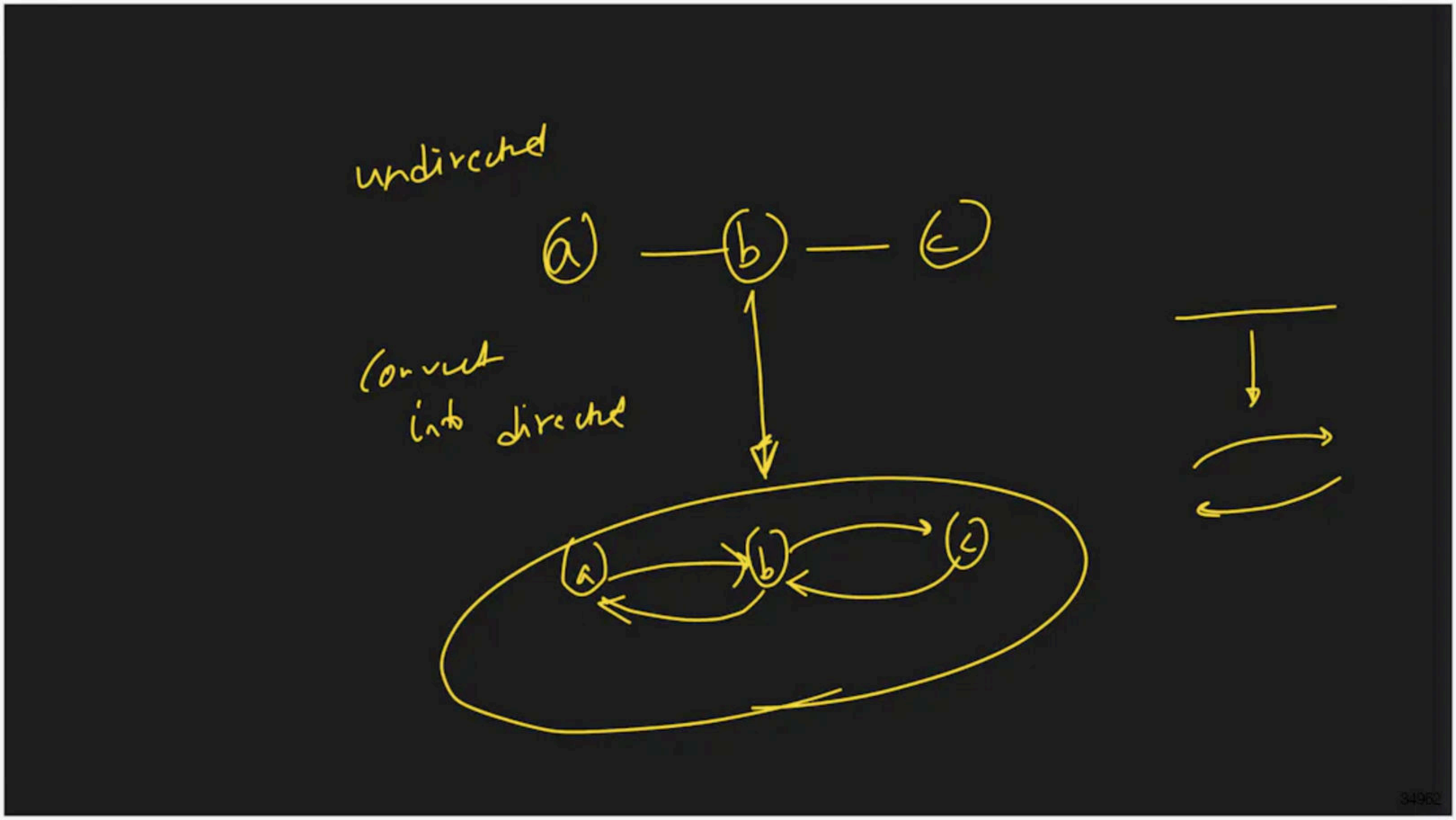
Graphs Class - 1

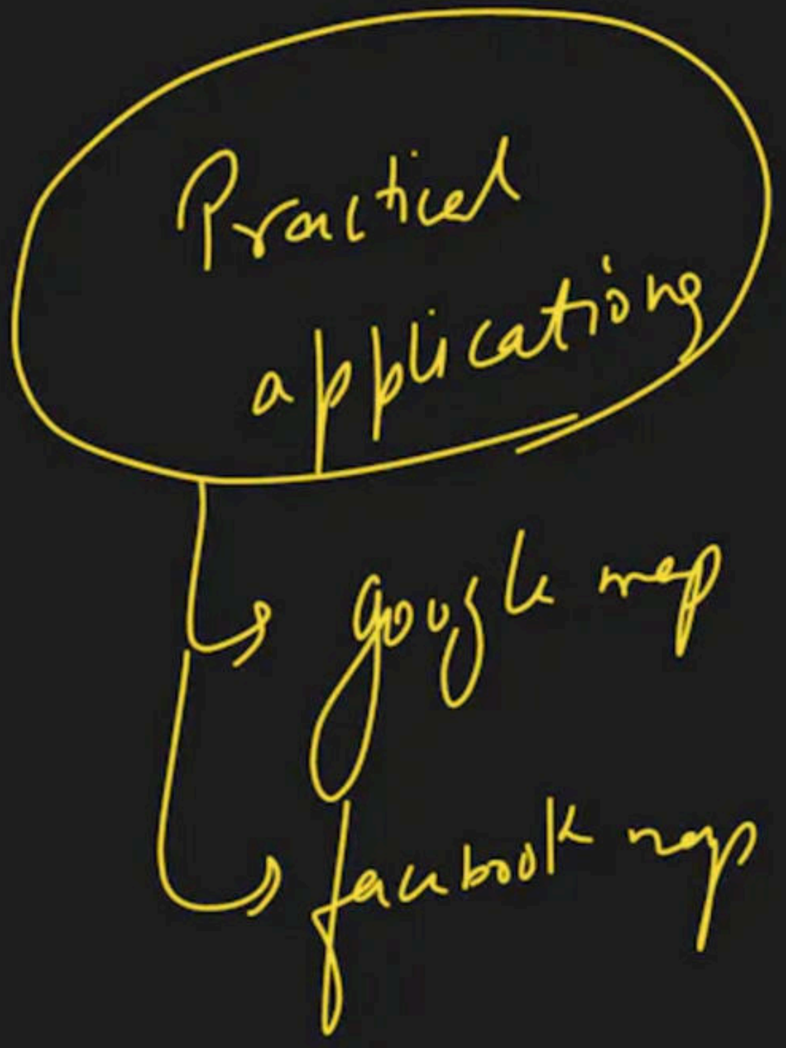
Special class

Love Babbar • Jan 24, 2024


Graph
D.S
Edges
Node
node
a
c
d
b
edge
int
char
node
d
graph
int
char
string
class


edge:-
directed
undirected
a → b
b → c
b → d
edge list
edge list


undirected
a
b
c
convert
into directed
a
b
c


Practical
applications
google map
facebook map

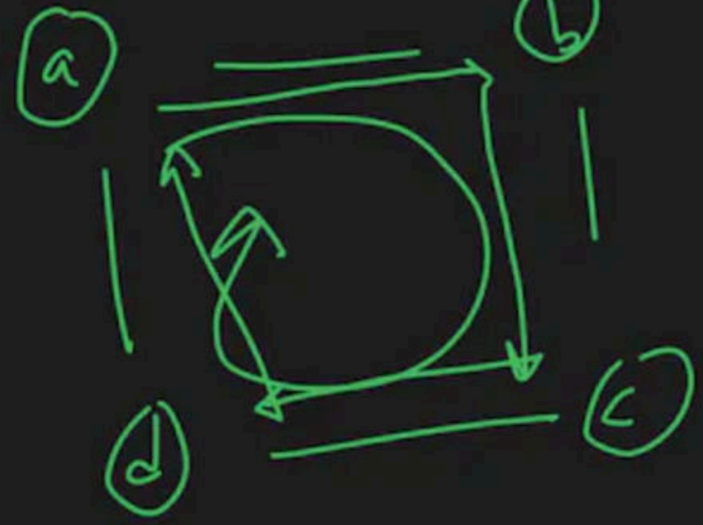
edge with no weight

unweighted edge

weight

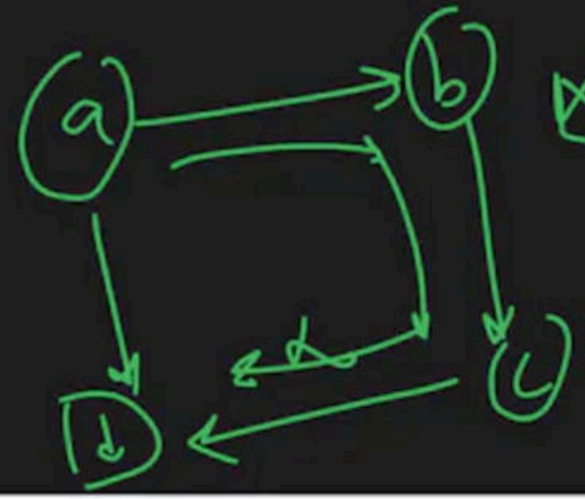
weighted graph

Cyclic

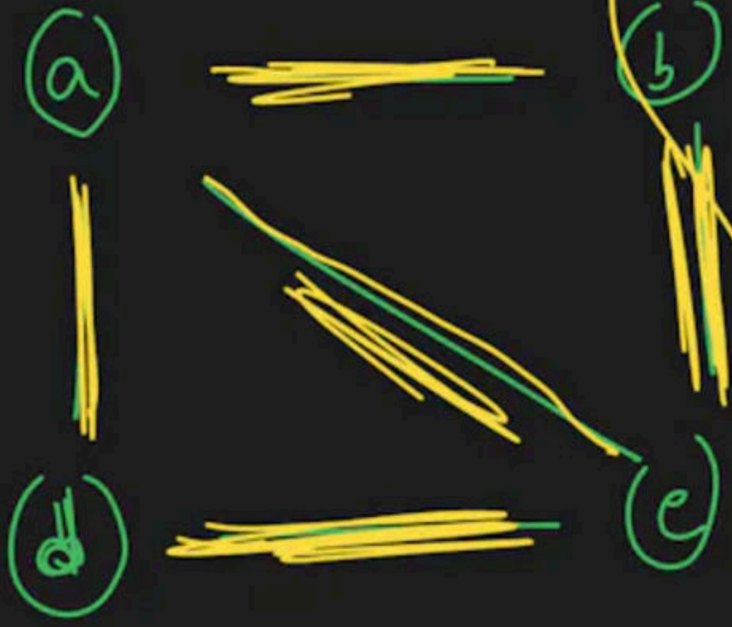


Cyclic graph

Acyclic



degree



degree(a) = 3

degree(b) = 2

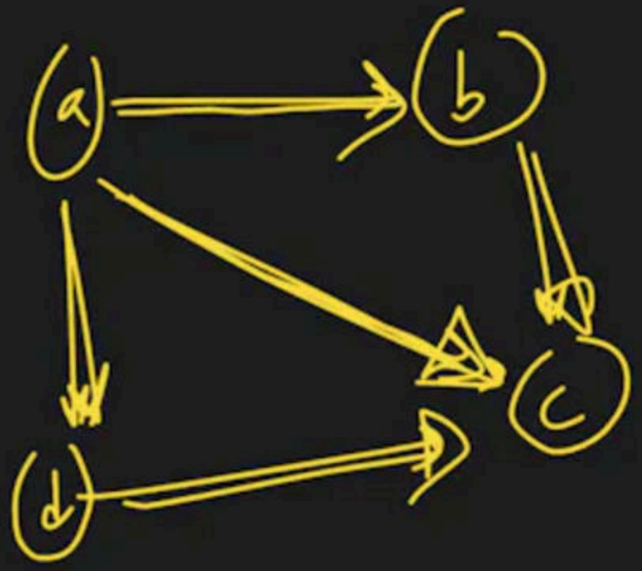
degree(d) = 2

degree(e) = 3

Directed graph

Indegree

Outdegree



Indegree(a) = 0

Outdegree(c) = 3

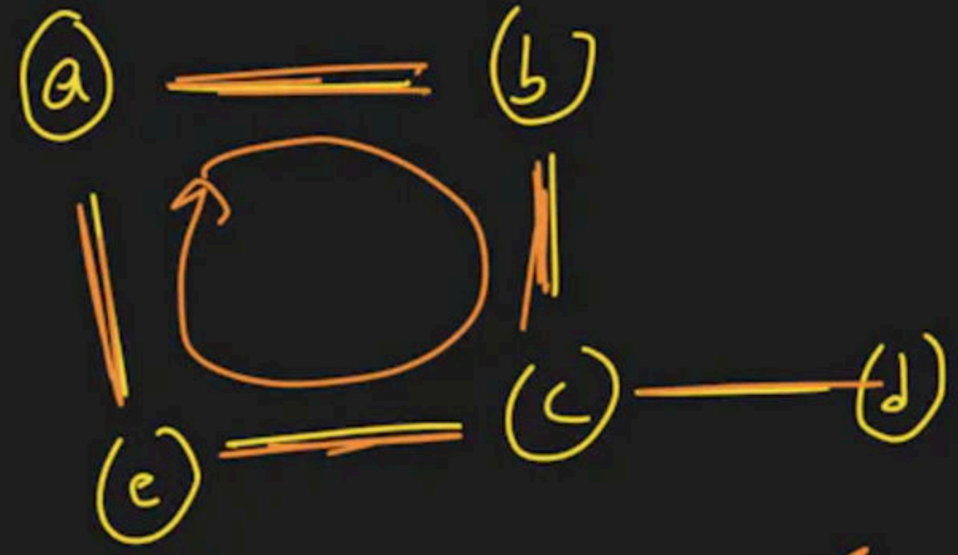
Indegree(c) = 2

Outdegree(c) = 0

61036

Path :-

a - b - c - c - c



a - b - c - d

a - b - c - e

graph

→ path

→ edge

→ node

→ undirected

→ directed

→ weighted

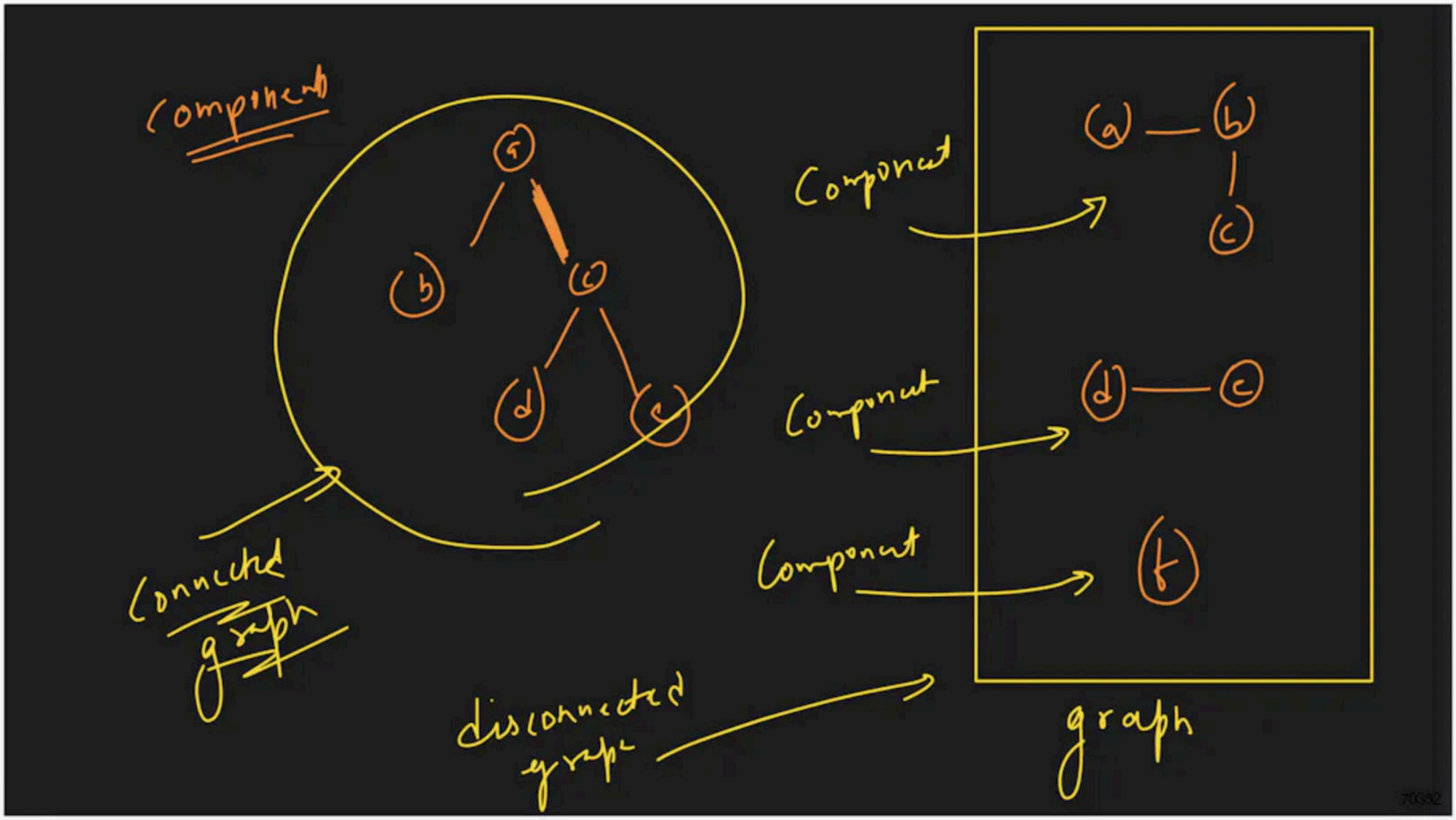
→ unweighted

→ degree

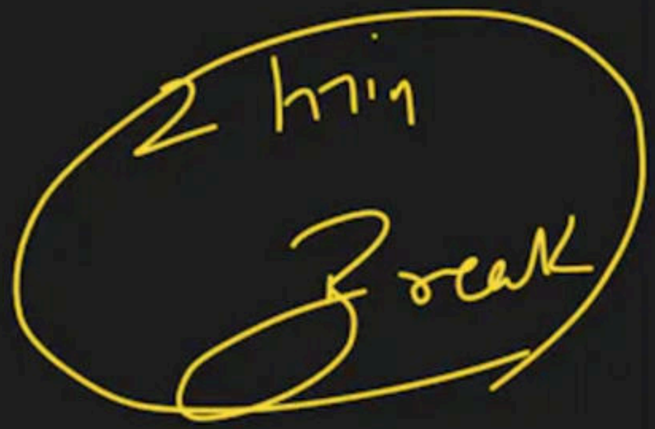
→ indegree

→ outdegree

→ cyclic / Acyclic

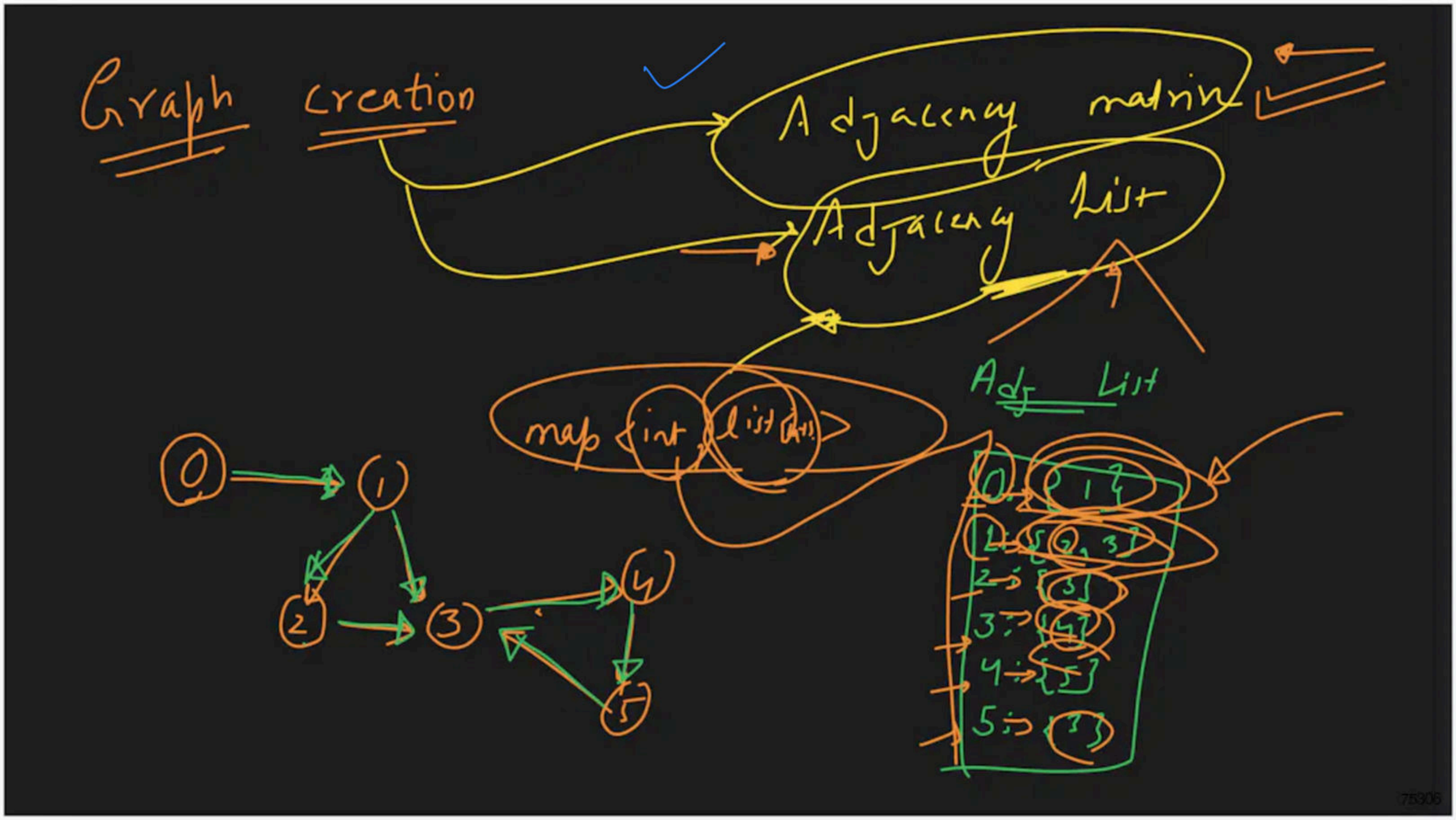

Components
a
b
c
d
e
Connected graph
disconnected graph
Component
Component
Component
graph

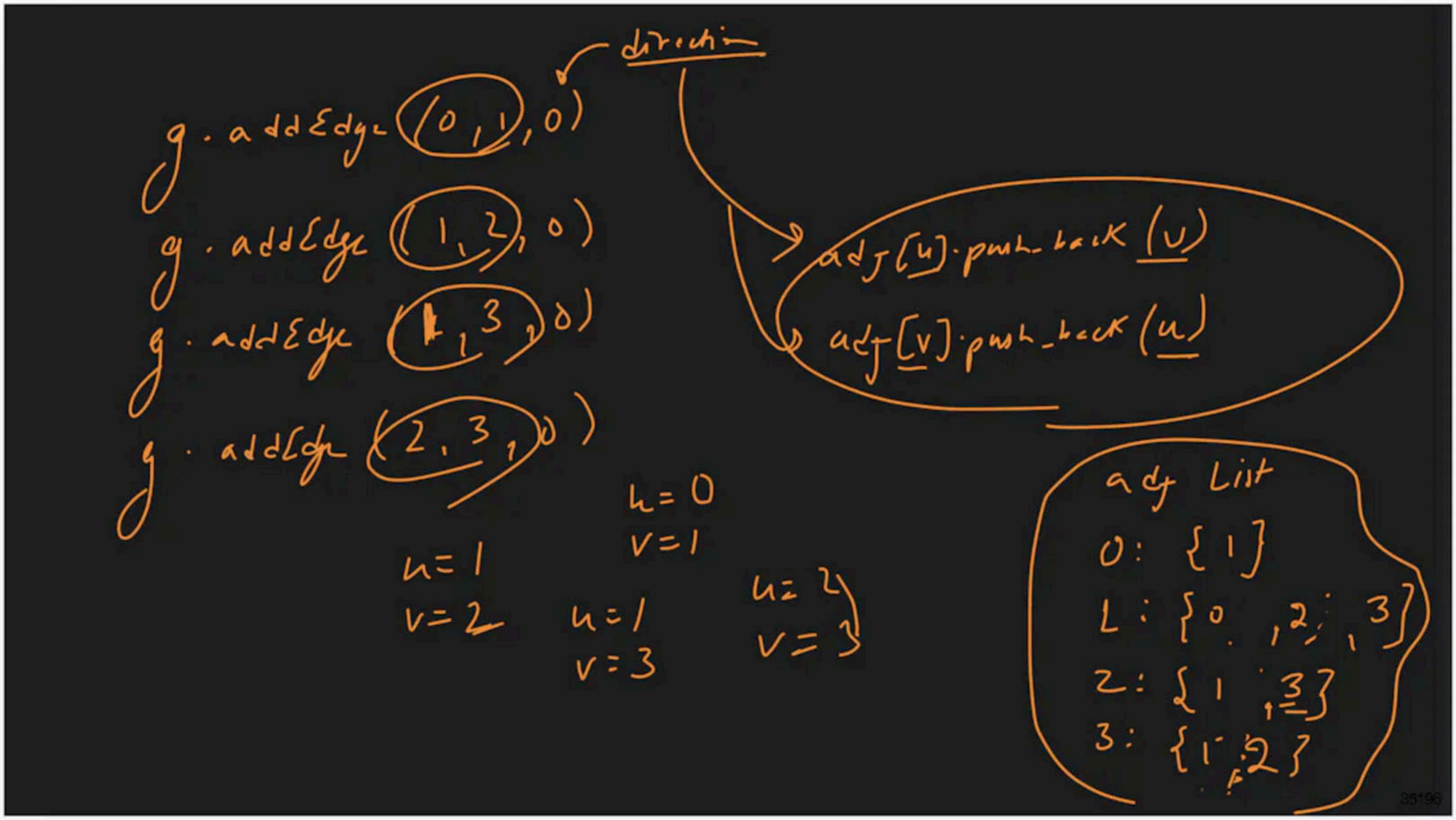

1
2
3

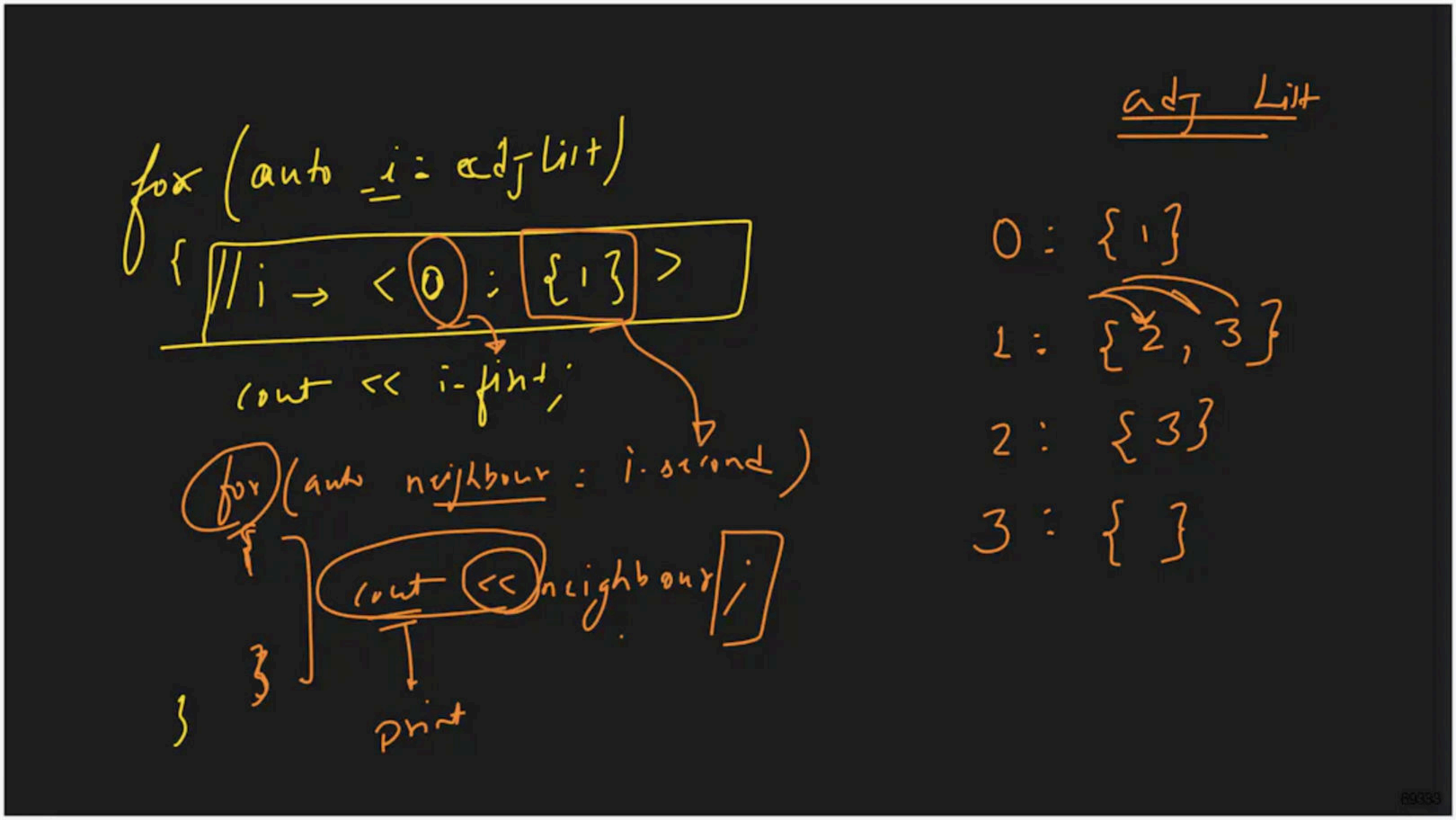

2 min
break


① clone a graph
②

Graph Creation

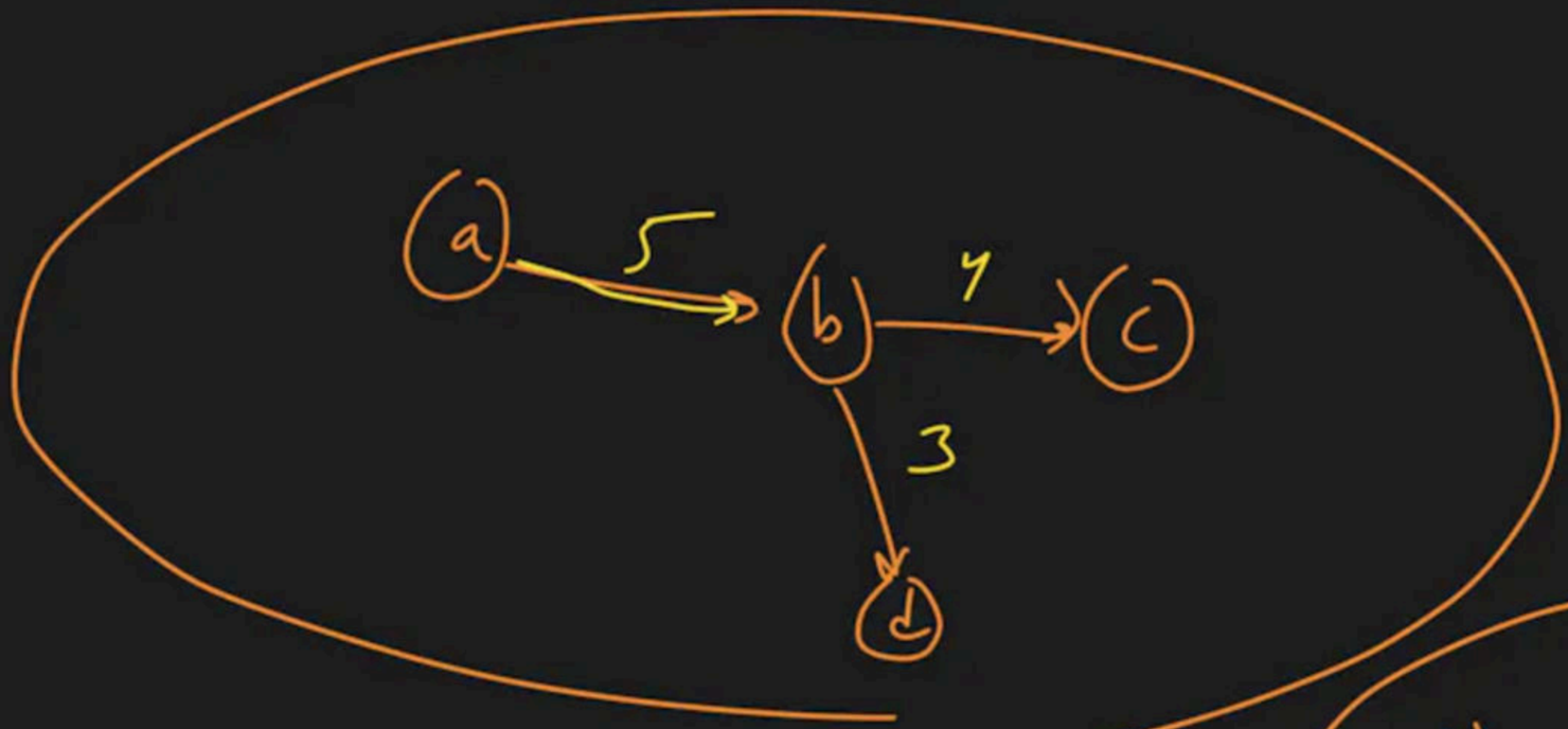



directed
g.addEdge(0, 1, 0)
g.addEdge(1, 2, 0)
g.addEdge(1, 3, 0)
g.addEdge(2, 3, 0)
adj[u].push_back(v)
adj[v].push_back(u)
u = 0
v = 1
u = 1
v = 2
u = 1
v = 3
u = 2
v = 3
adj List
0: {1}
1: {0, 2, 3}
2: {1, 3}
3: {1, 2}


adj List
for (auto _i : adj List)
{
// i → < 0 : { 1 } >
cout << i.first;
for (auto neighbour : i.second)
{
cout << neighbour;
}
}
print
0 : { 1 }
1 : { 2, 3 }
2 : { 3 }
3 : { }

Rawat

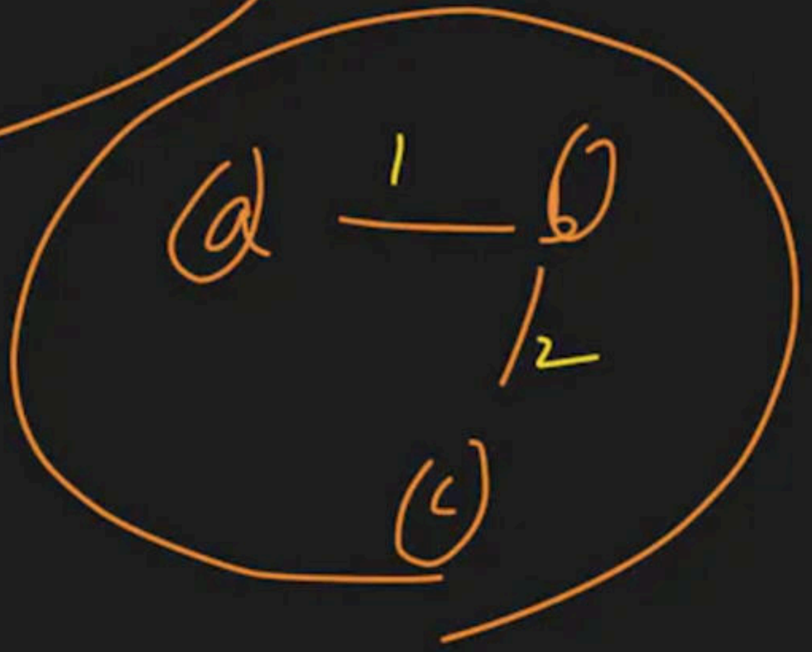
weighted

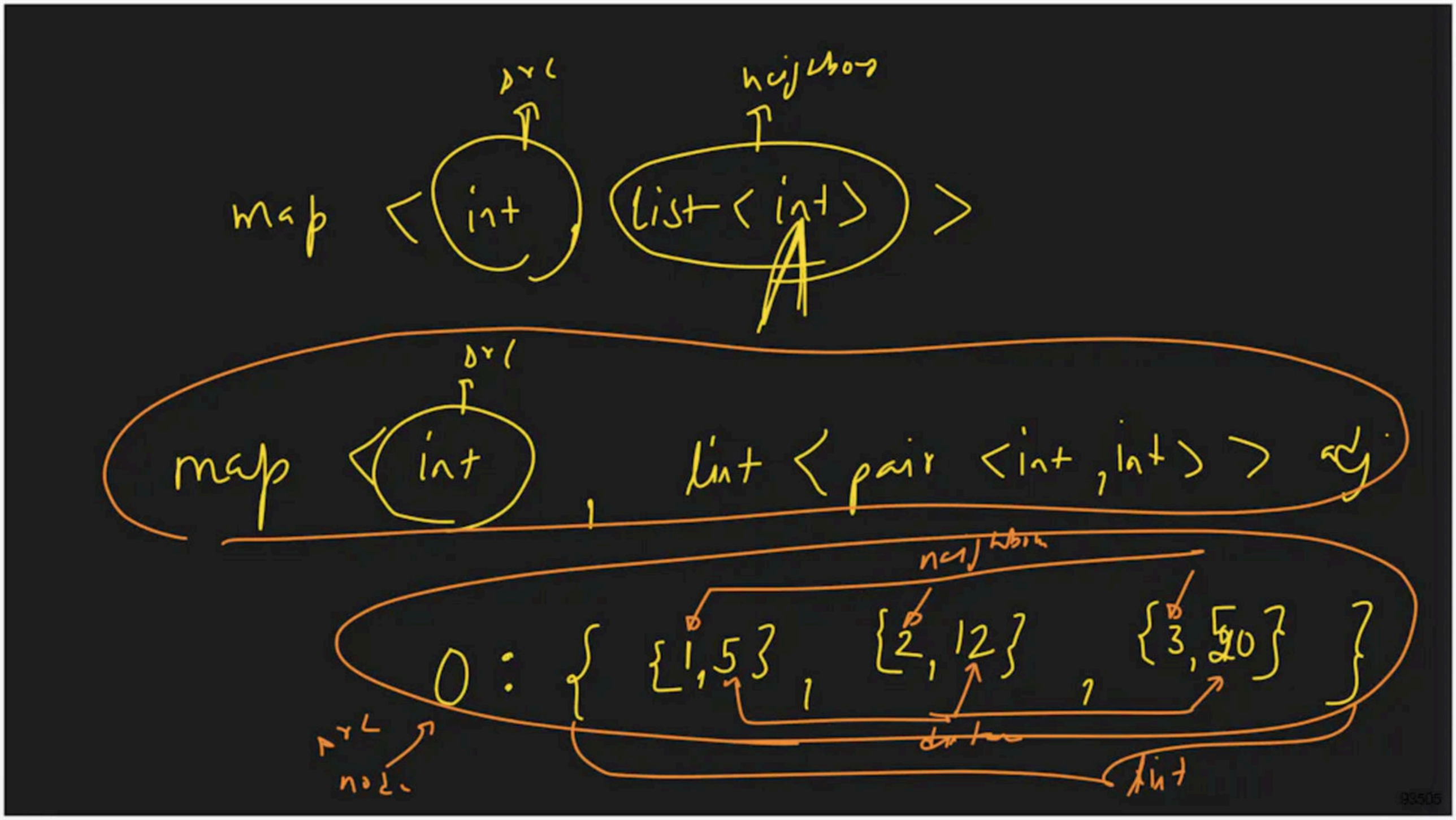


a —5→ b

b —7→ c

b —3→ d




src
neighbors
map < int list<int> >
src
map < int list<pair <int, int> > adj
neighbors
0 : { {1, 5} {2, 12} {3, 20} }
src
node
dist
int

adj list

E = 8

V = 4 ⇒ 12

0 : { 1, 2 } 2
1 : { 0, 3 } 2
2 : { 0 } 2
3 : { 2, 1 } 2

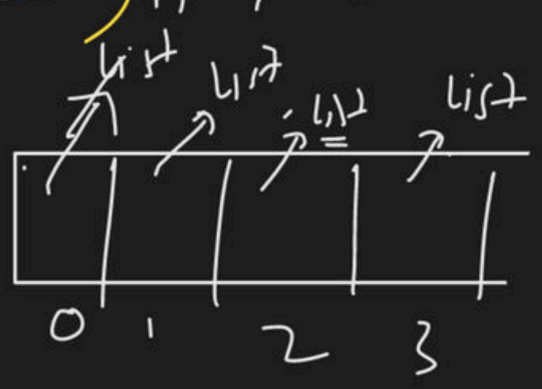
V → no. of vertices

E → edges

S.C: O(V+E)

T.C: O(V + E)

8

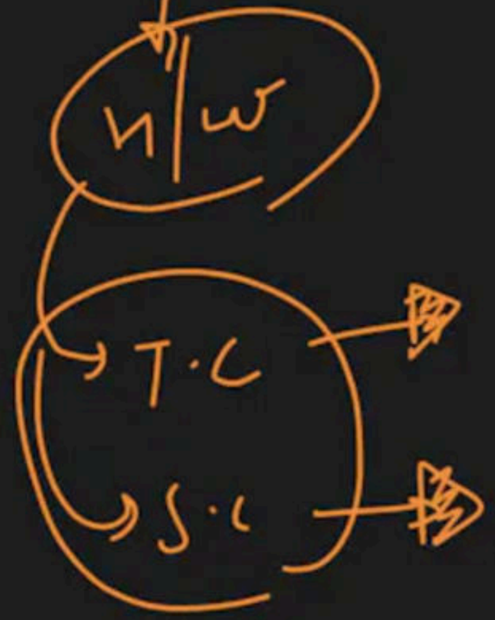


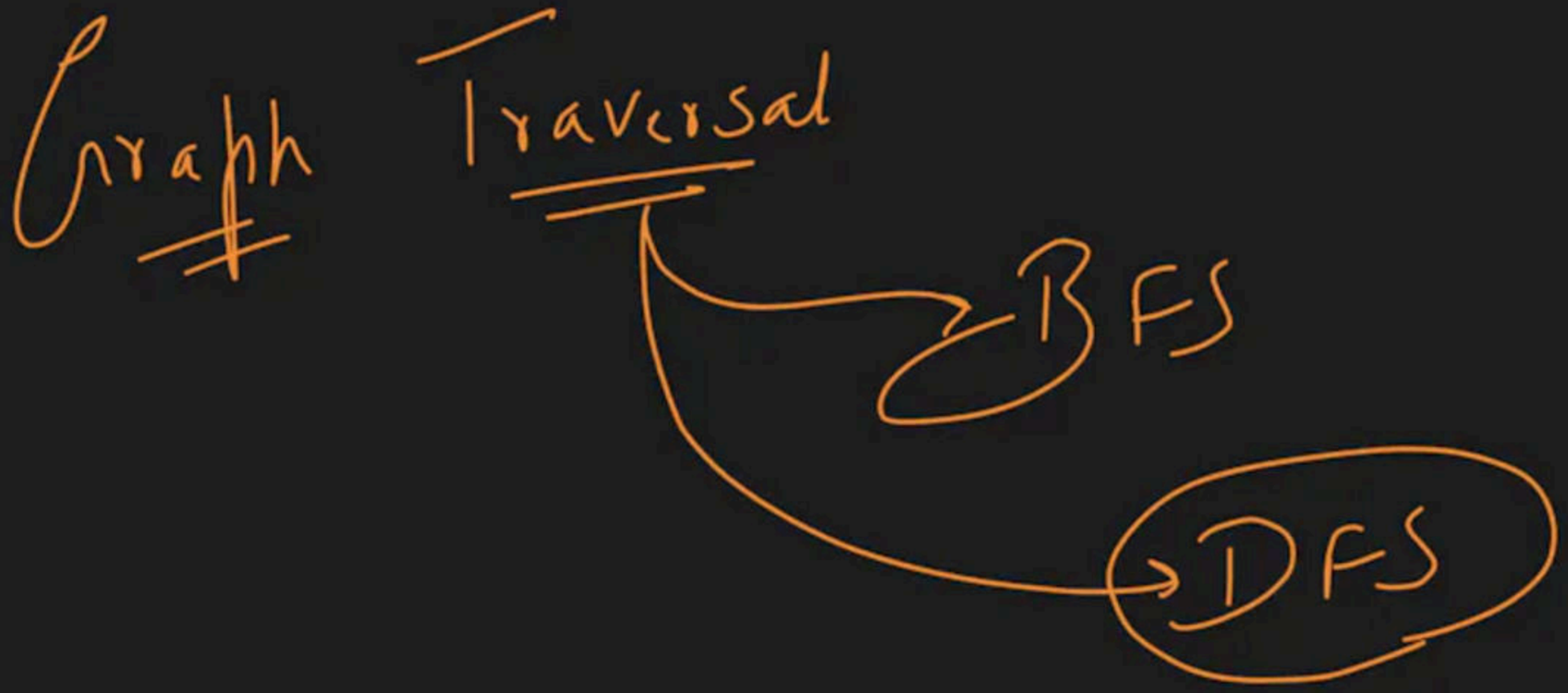
① int V

② Edge

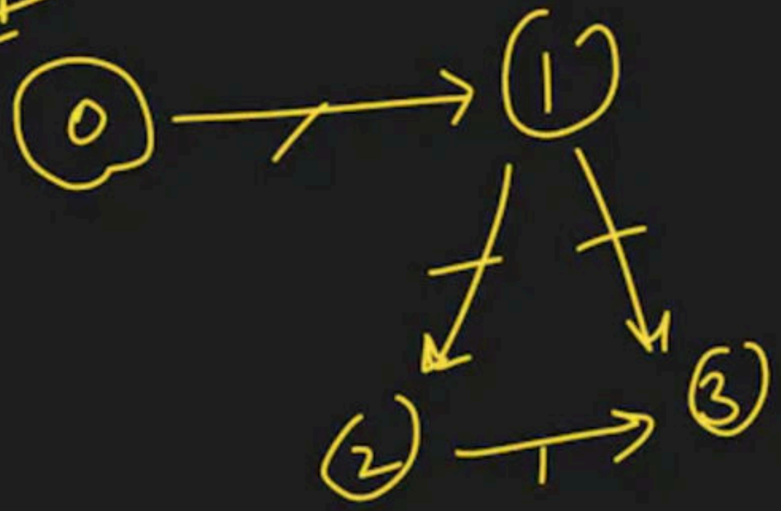
graph

↳ adj List

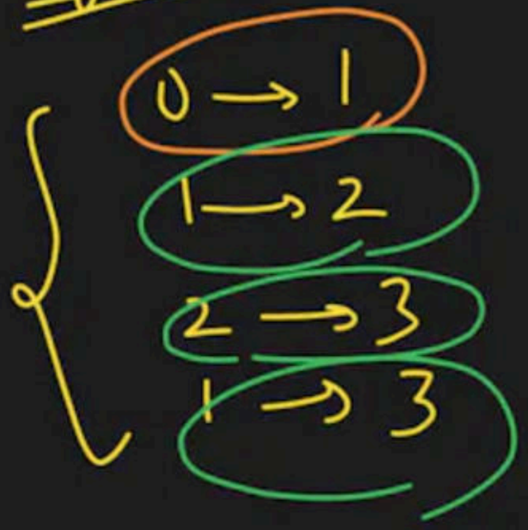



Graph
Traversal
BFS
DFS

graph



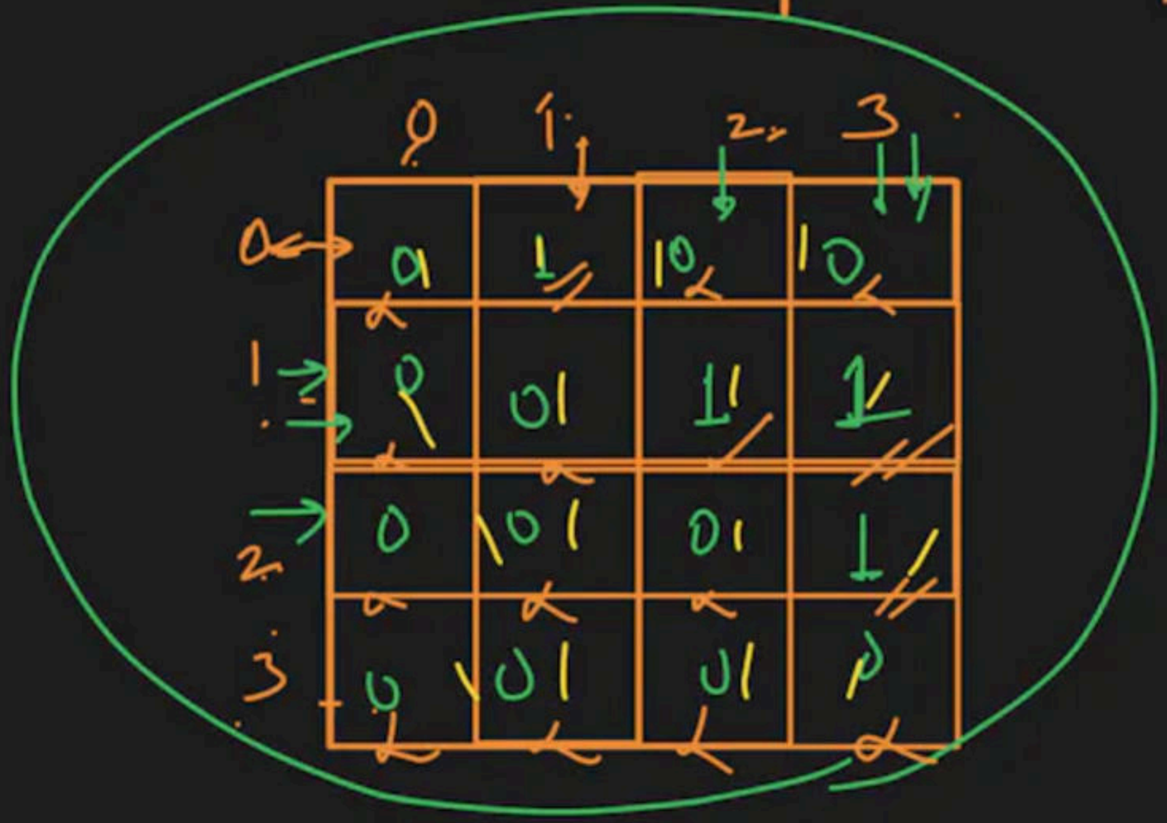
edge list



Adjacency matrix

n x n

no of node = 4 node



T.C → $O(n^2)$

```
void solve ( vector<pair<int,int>> edgelist )
{
    int n = edgelist.size
    vector<vector<int>> adj (n, vector<int>(n,0));
    for ( auto i : edgelist )
    {
        int u = i.first;   int v = i.second;
        adj[u][v] = 1;
    }
}
```

i → pair <int, int>

u

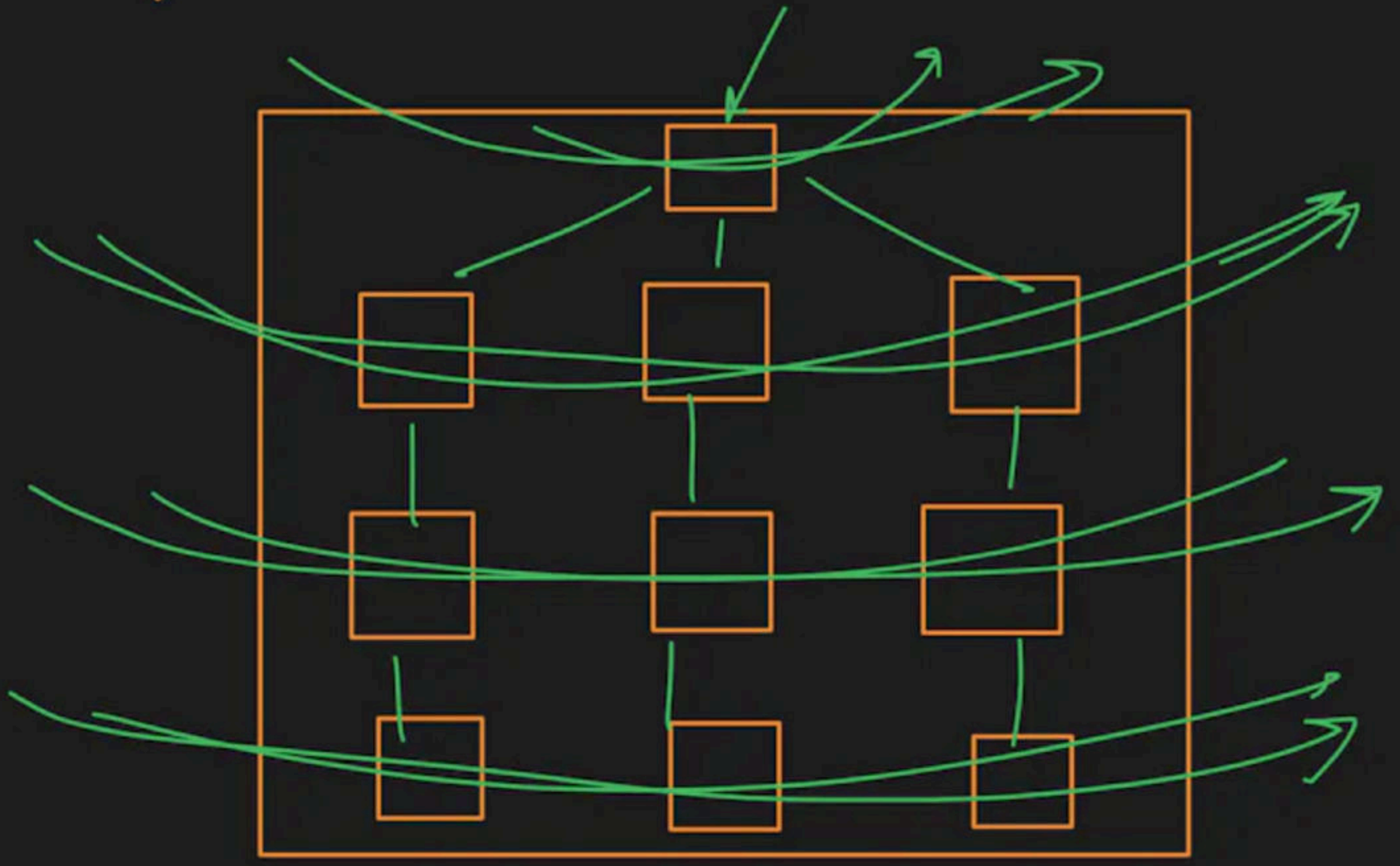
v

u → v

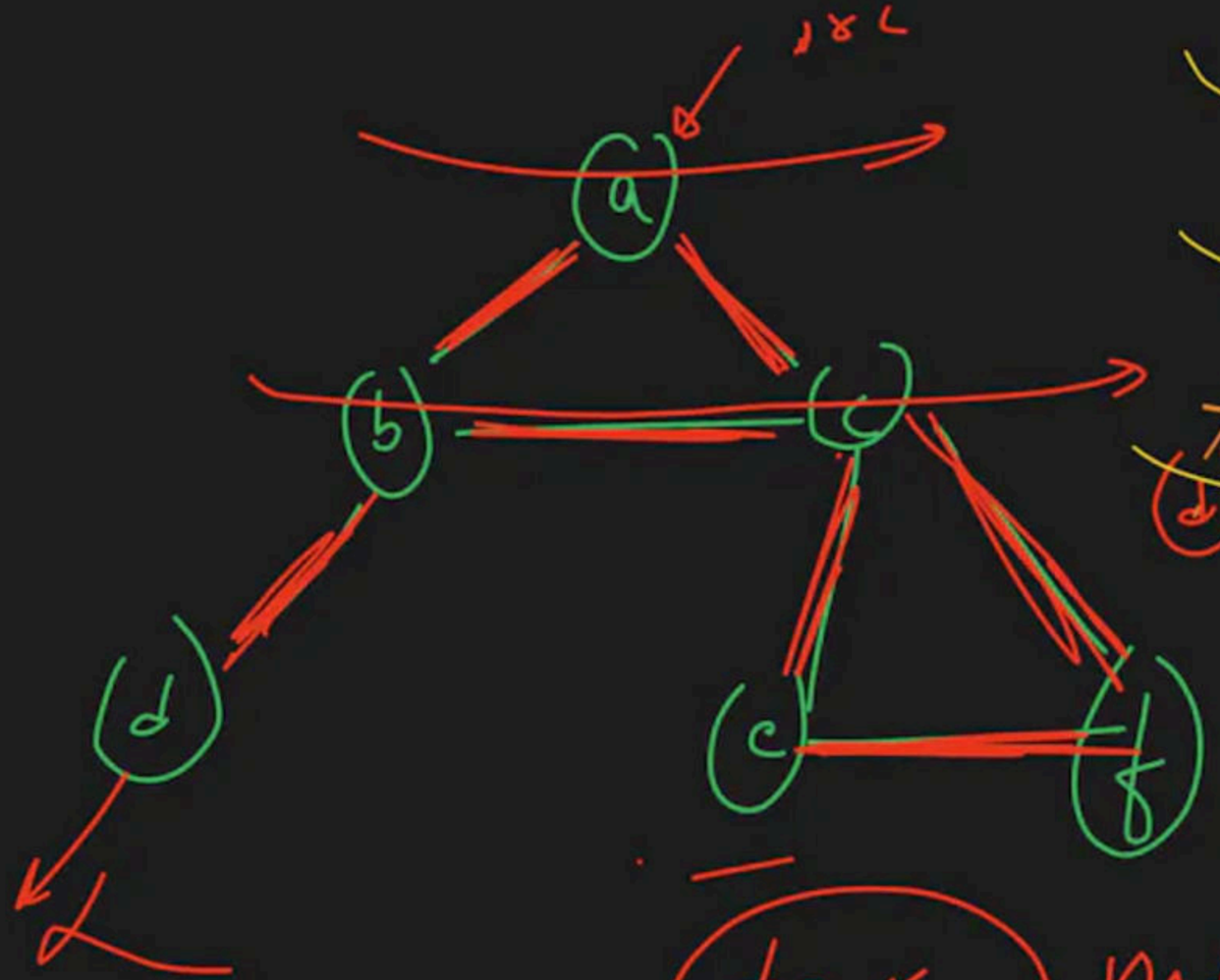
S.C → $O(n^2)$

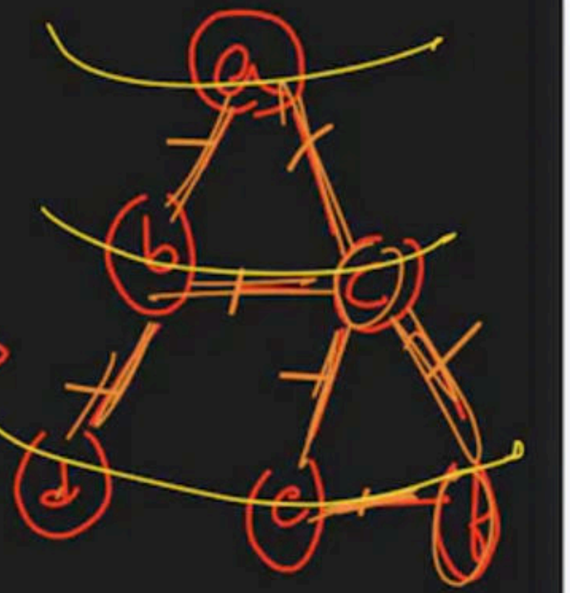
BFS :- Breadth first Search

queue



a
b c
d e f





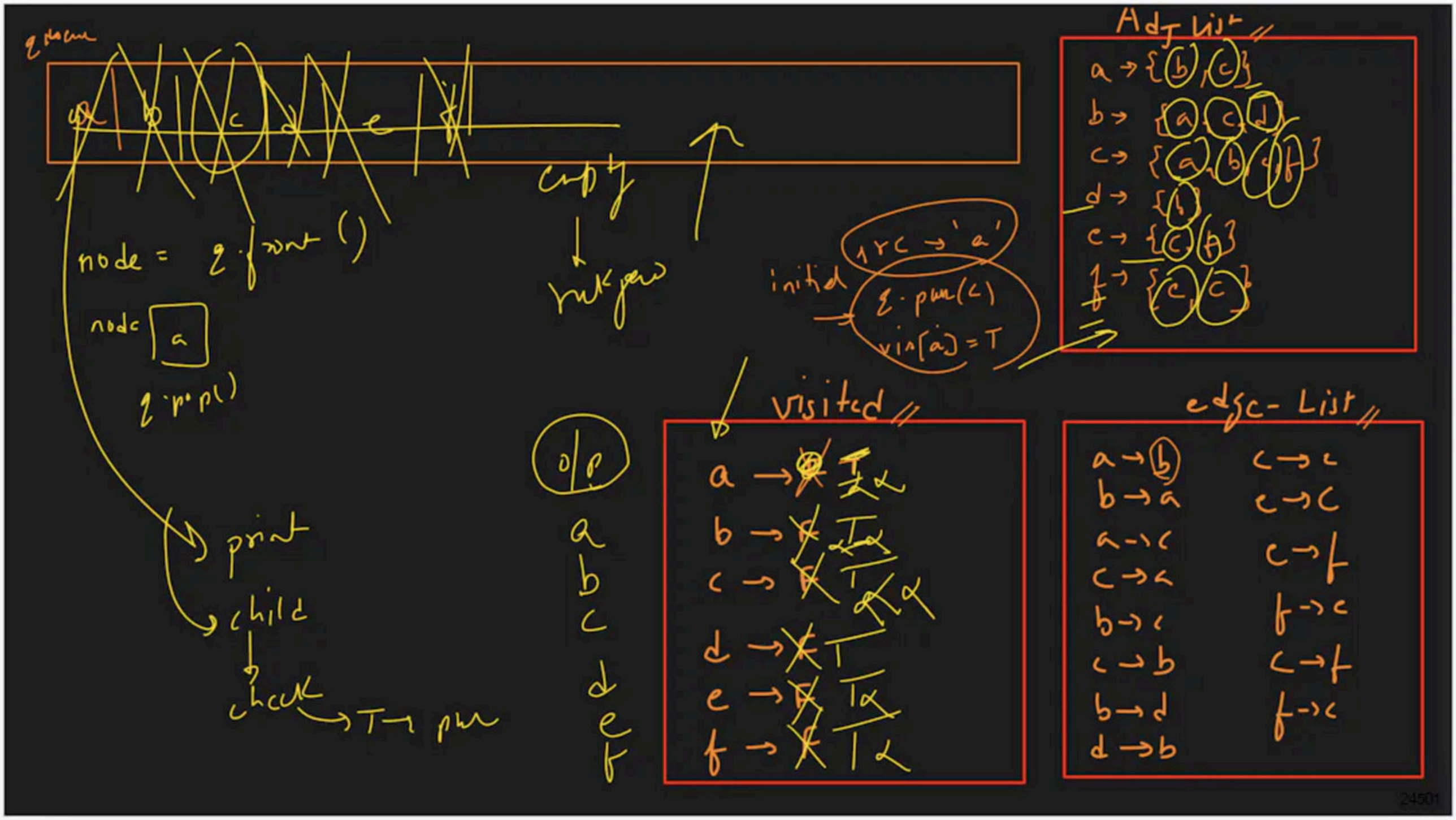
track

node

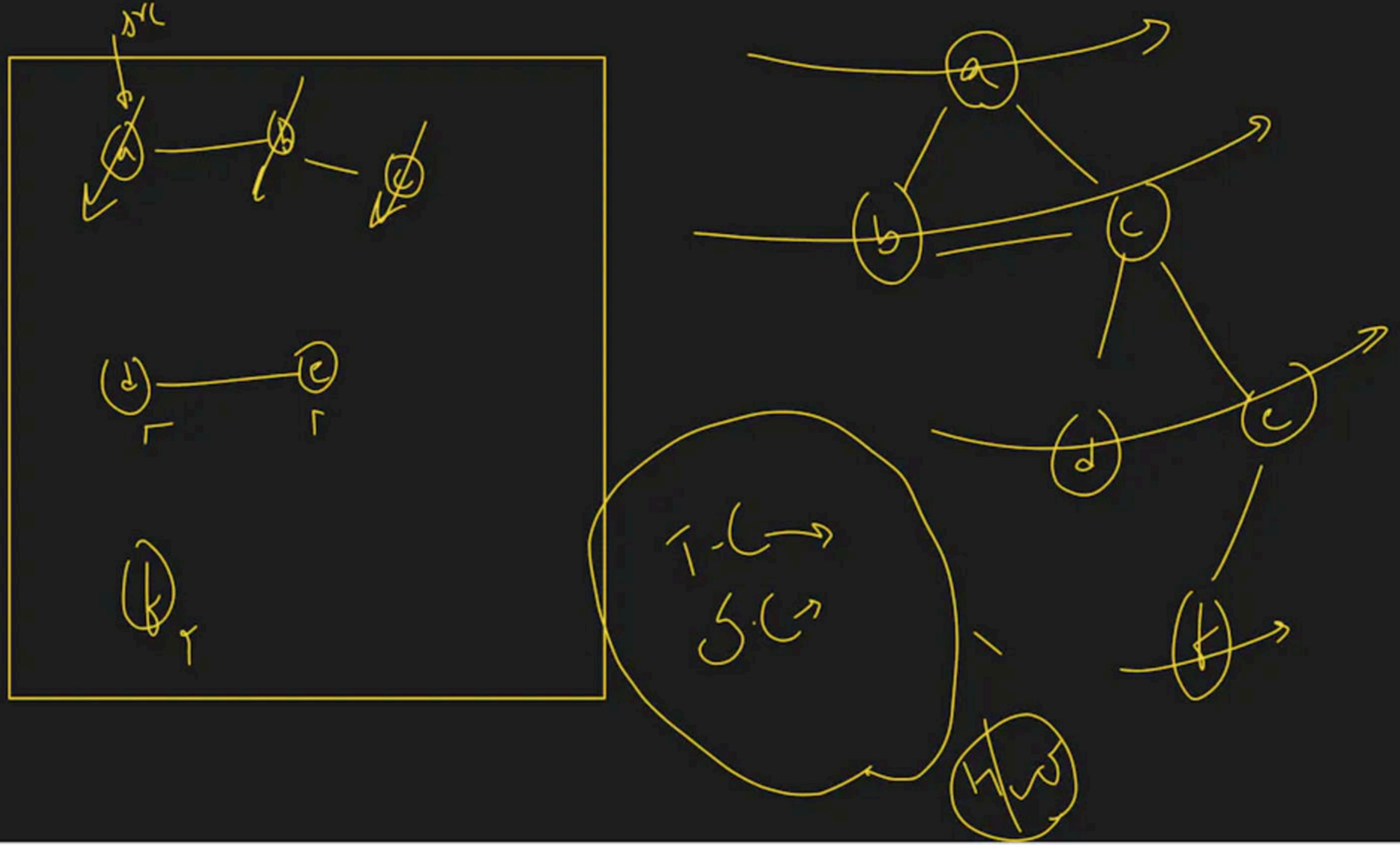
visited

! visited

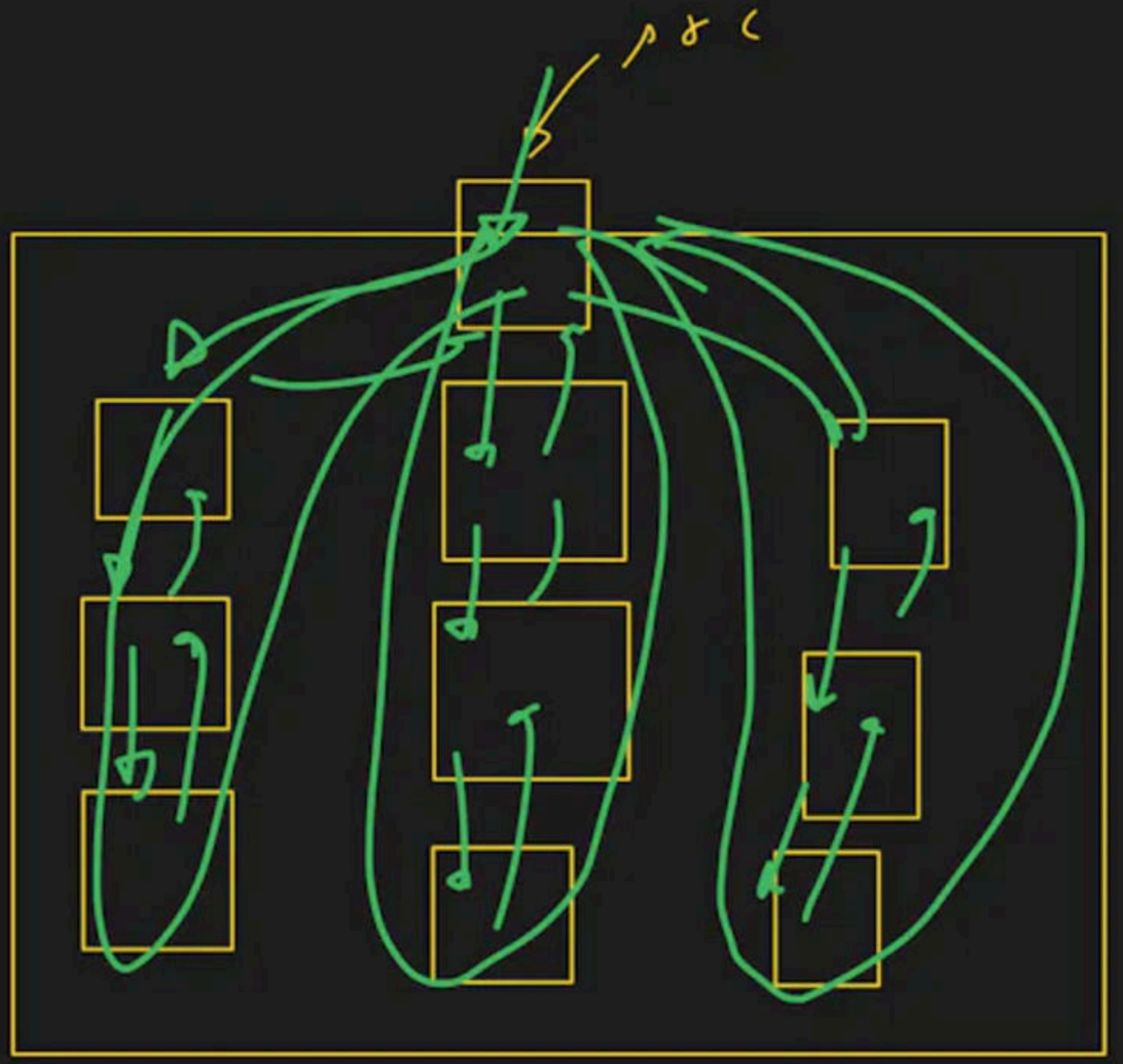
DFS → T/F

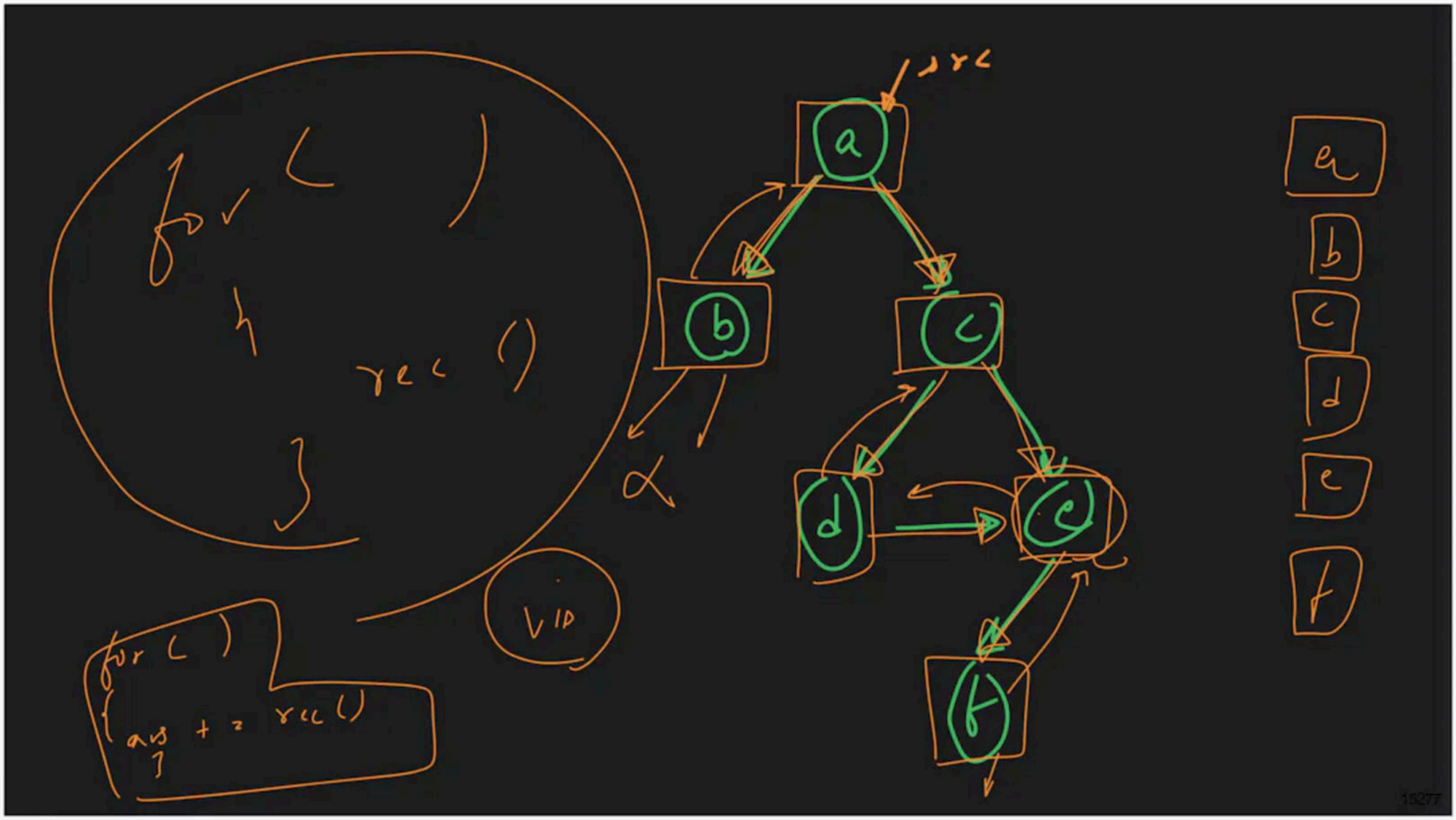

queue
a
b
c
e
f
node = q.front()
node
a
q.pop()
copy
neighbours
print
child
check
T → push
o/p
a
b
c
d
e
f
visited
Adj List
a → {b, c}
b → {a, c, d}
c → {a, b, e, f}
d → {b}
e → {c, f}
f → {c, e}
src → 'a'
initial
q.push(c)
vis[a] = T
edge - List
a → b
b → a
a → c
c → a
b → e
e → b
b → d
d → b
c → e
e → c
c → c
c → f
f → e
e → f
f → c

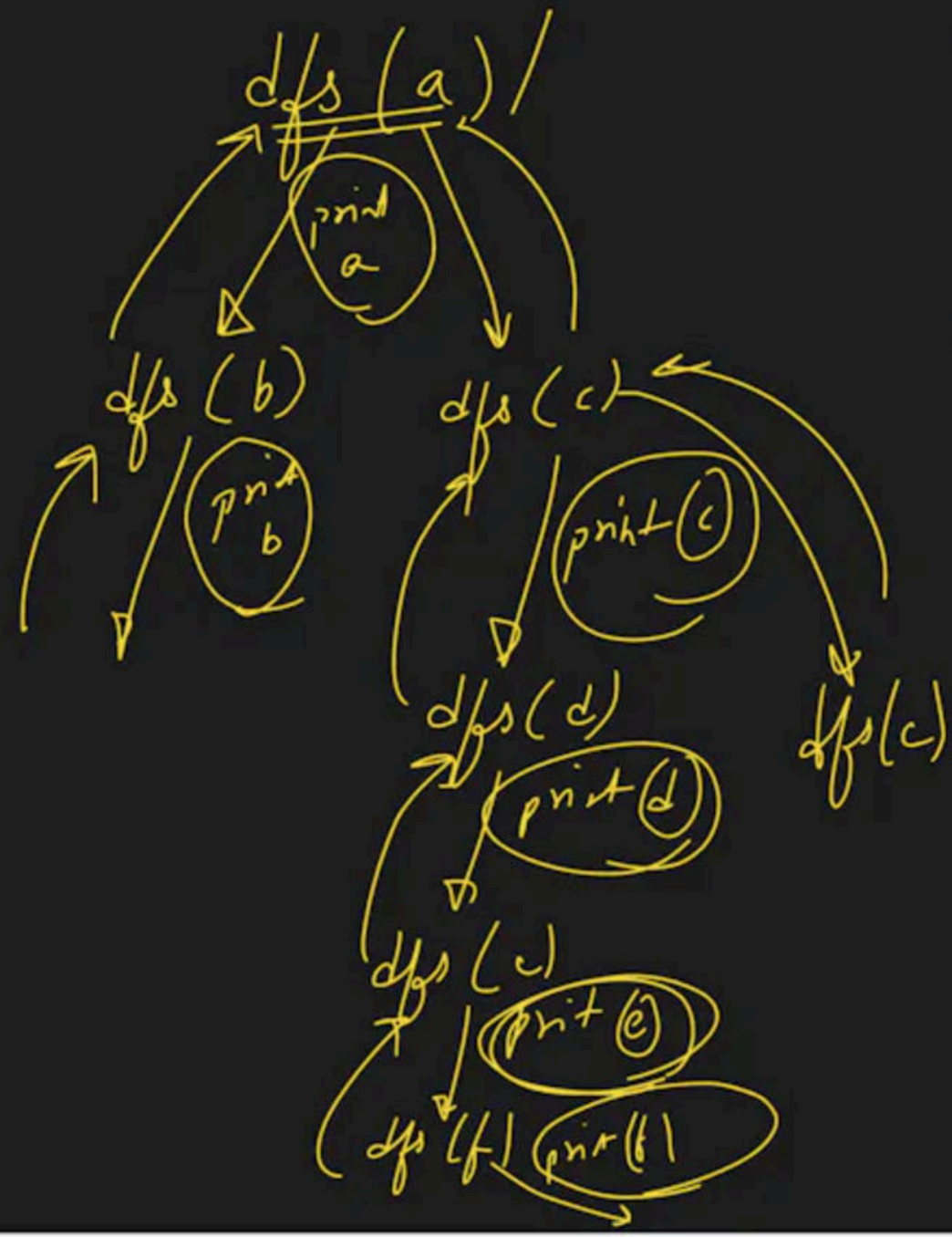
24501

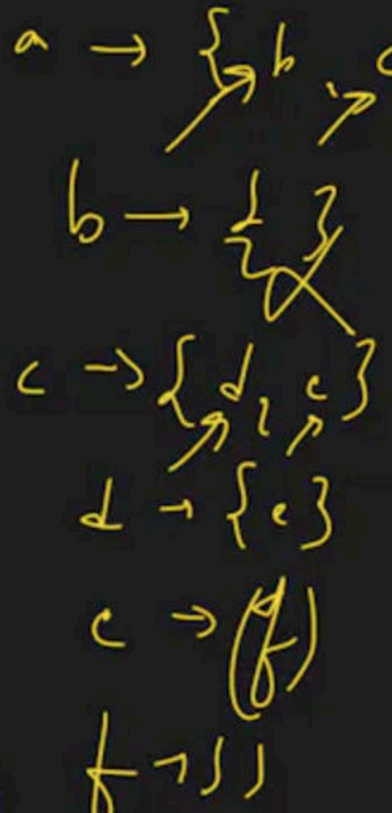


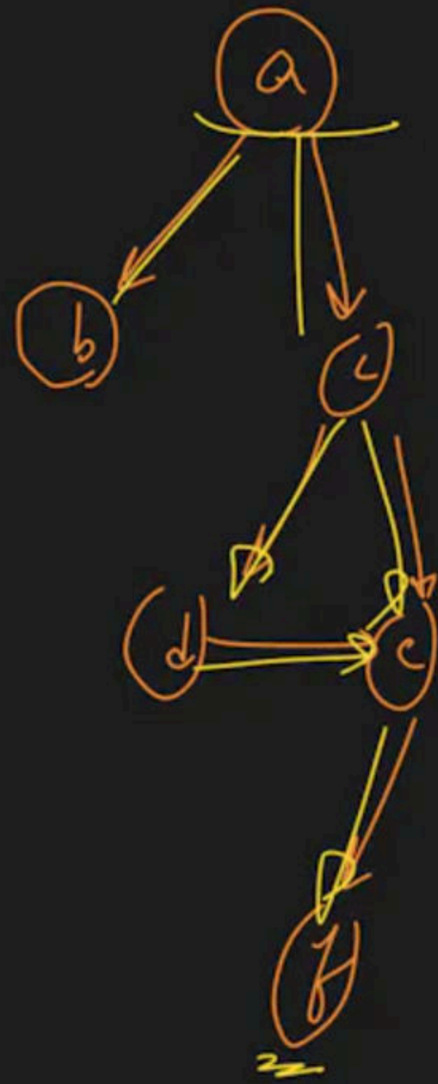
DFS

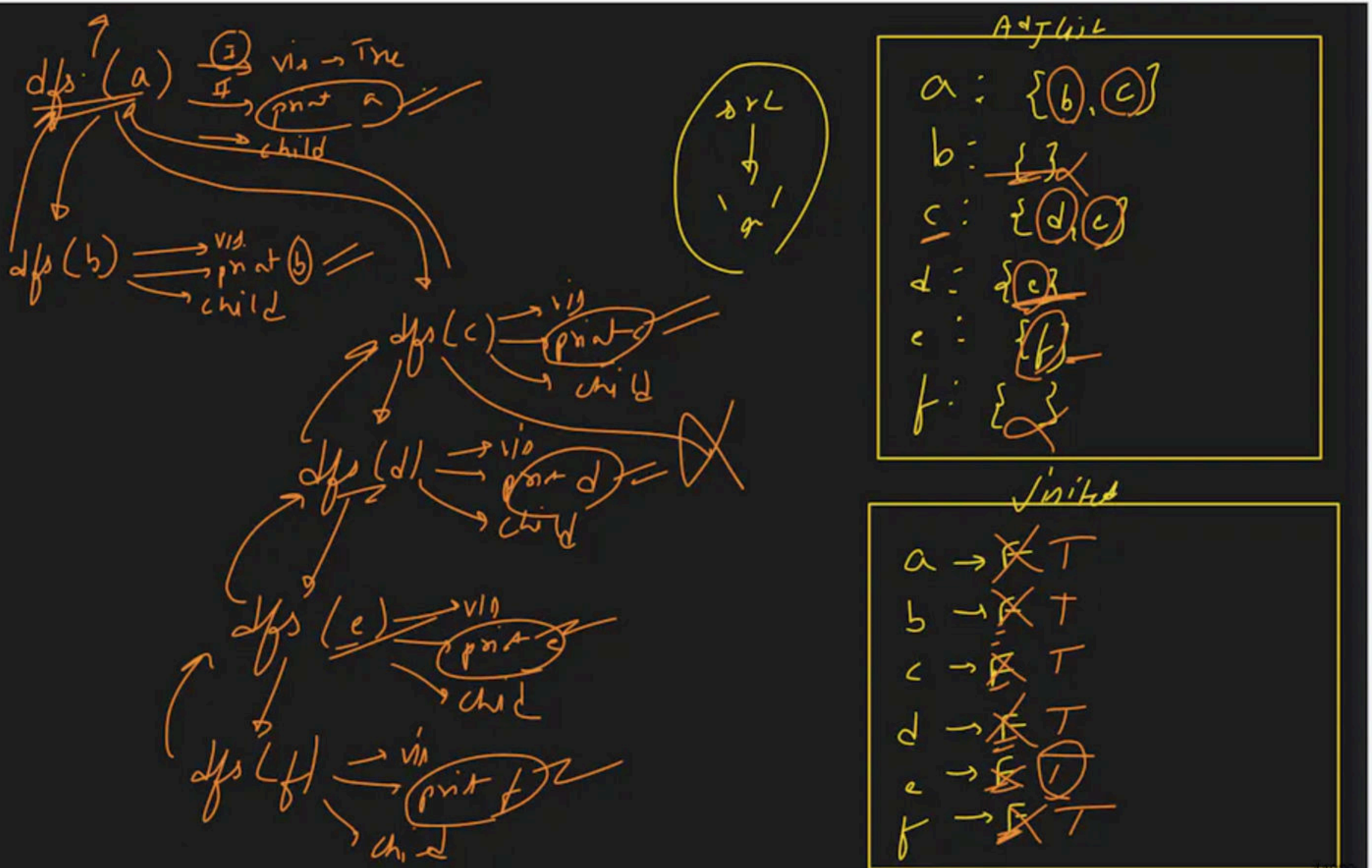


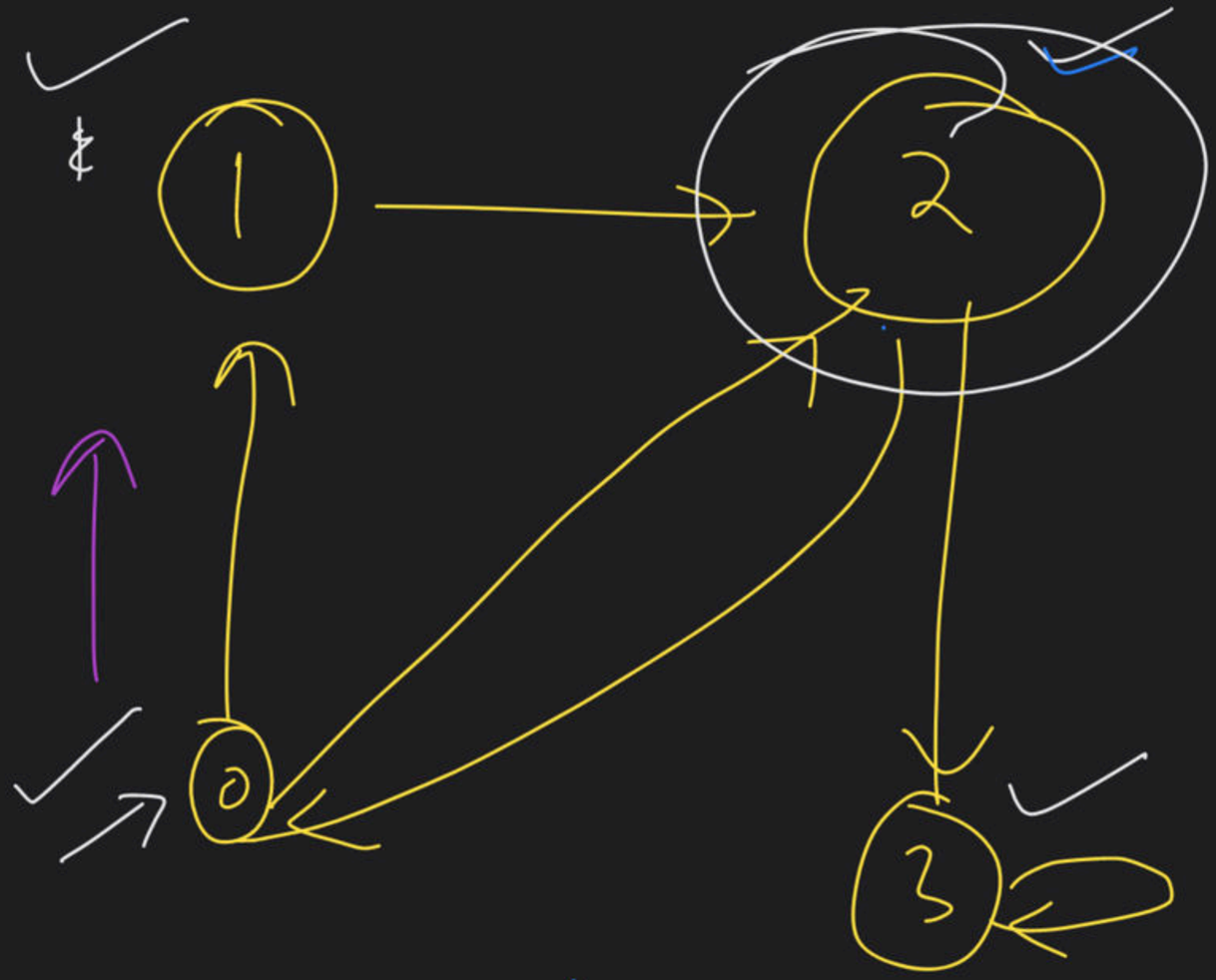

for ( )
{
rec ()
}
VID
for ( )
{
ans += rec()
}
a
b
c
d
e
f
a
b
c
d
e
f


dfs (a)
print a
dfs (b)
print b
dfs (c)
print c
dfs (d)
print d
dfs (e)
print e
dfs (f)
print f
dfs (c)


a → {b, c}
b → {}
c → {d, e}
d → {e}
e → {f}
f → {}


a
b
c
d
e
f


dfs(a)
vis → True
print a
child
dfs(b)
vis
print b
child
dfs(c)
vis
print c
child
dfs(d)
vis
print d
child
dfs(e)
vis
print e
child
dfs(f)
vis
print f
child
src
b
a
Adjlist
a: {b, c}
b: { }
c: {d, e}
d: {c}
e: {f}
f: { }
visited
a → F T
b → F T
c → F T
d → F T
e → F T
f → F T



$\Rightarrow$

dfs(0) → T

dfs(1)

dfs(2)

dfs(3)

if (N <= 0)

return 0

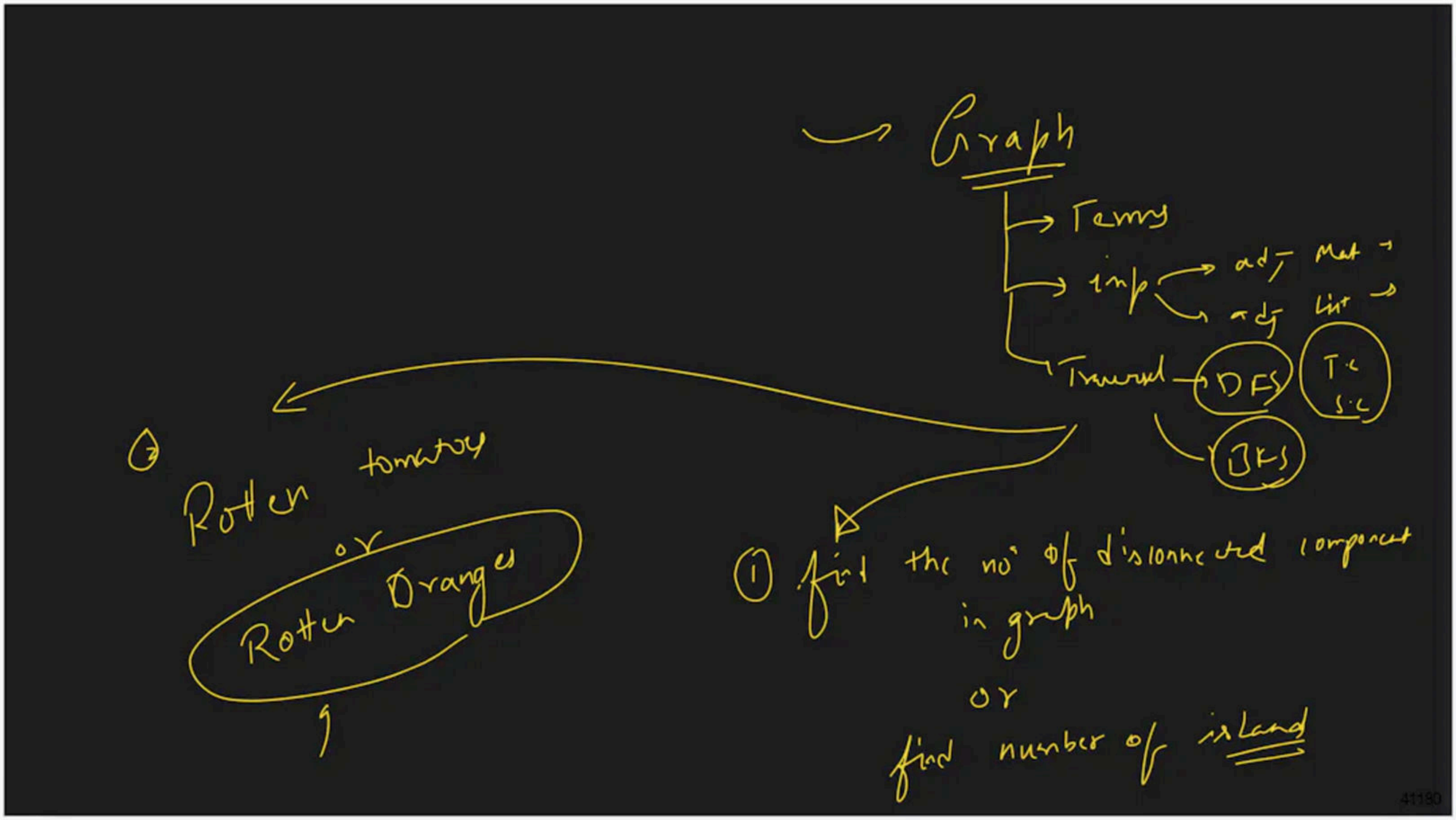
dfs( )

{ if (N > 0)

Graph Class - 2

Special class

Love Babbar • Jan 26, 2024


Graph
Terms
inp
adj Mat
adj List
Traversal
DFS
T.C
S.C
BFS
Rotten tomatoy
or
Rotten Oranges
find the no of disconnected component in graph
or
find number of islands

```
for(i ∈ [0, V-1])
{
  if(visited[i]==0)
    { dfs(i) }
}
```

DFS chalaa kee nahi

i=0 : ✓

i=1 = ✓

i=2 = X

i=3 = ✓

i=4 = ✓ (3)

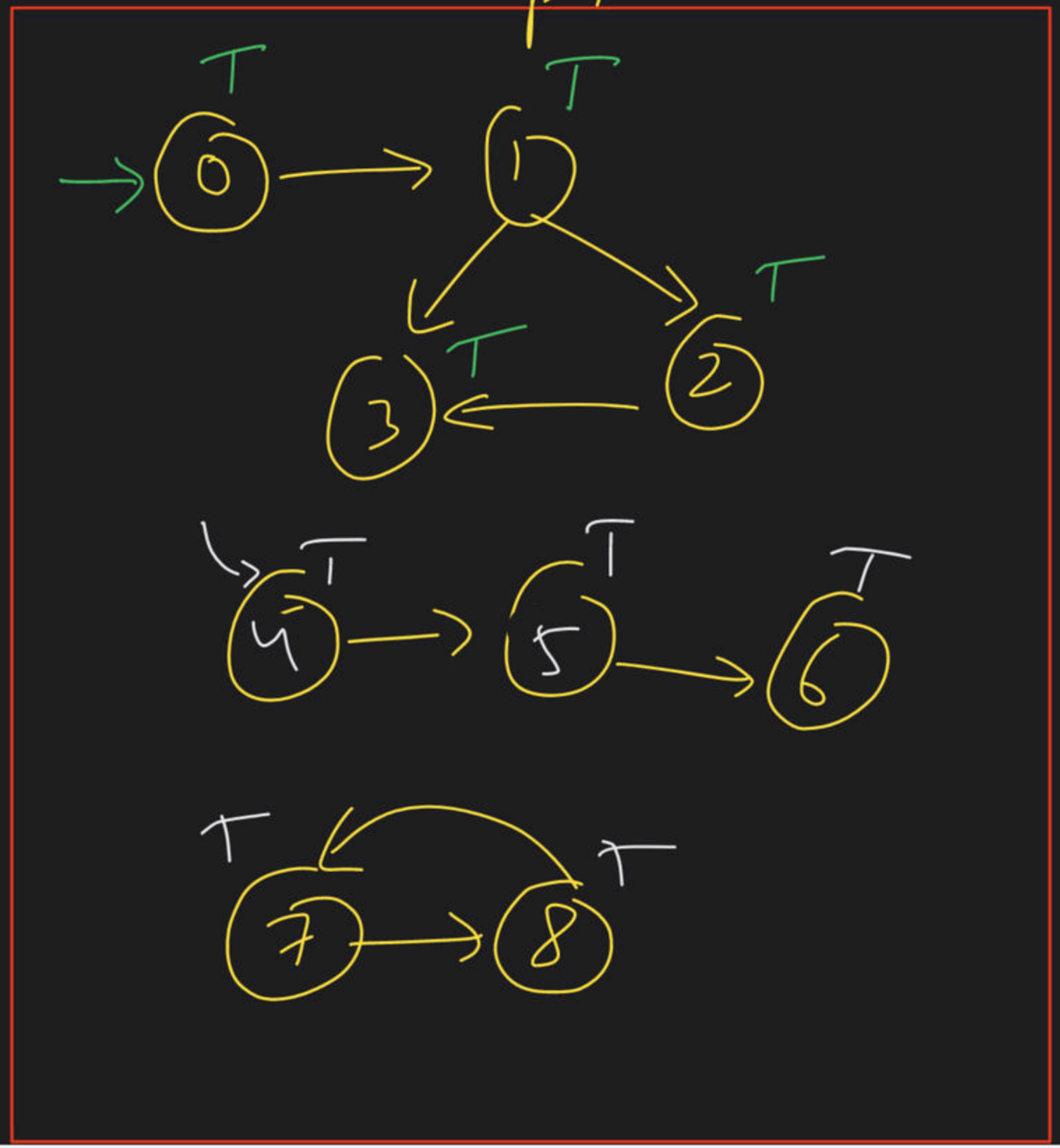
i=5 X

i=6 X ✓

7

8 = X

Graph



Visited map

0: T

1: T

2: T

3: T

4: T

5: T

6: T

7: T

8: T

H.W

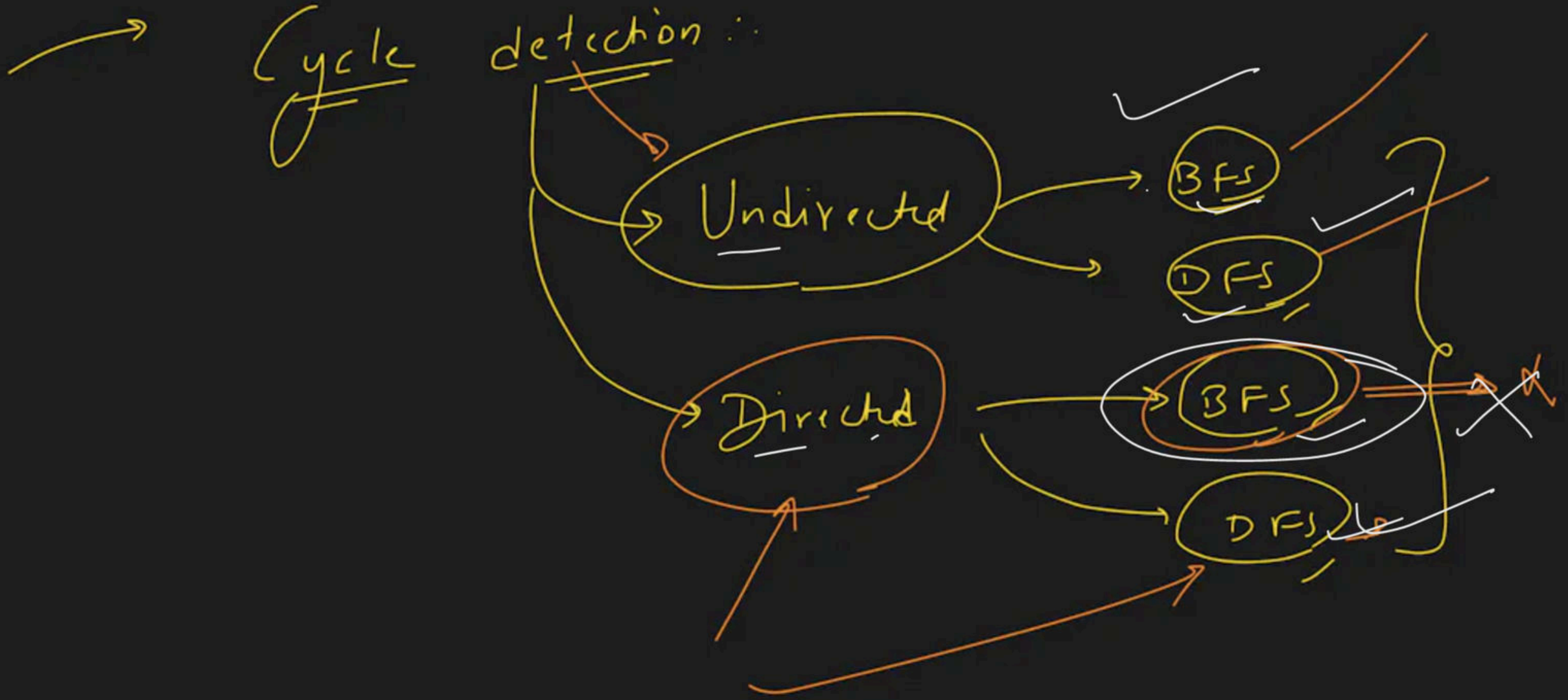
LC = 1319

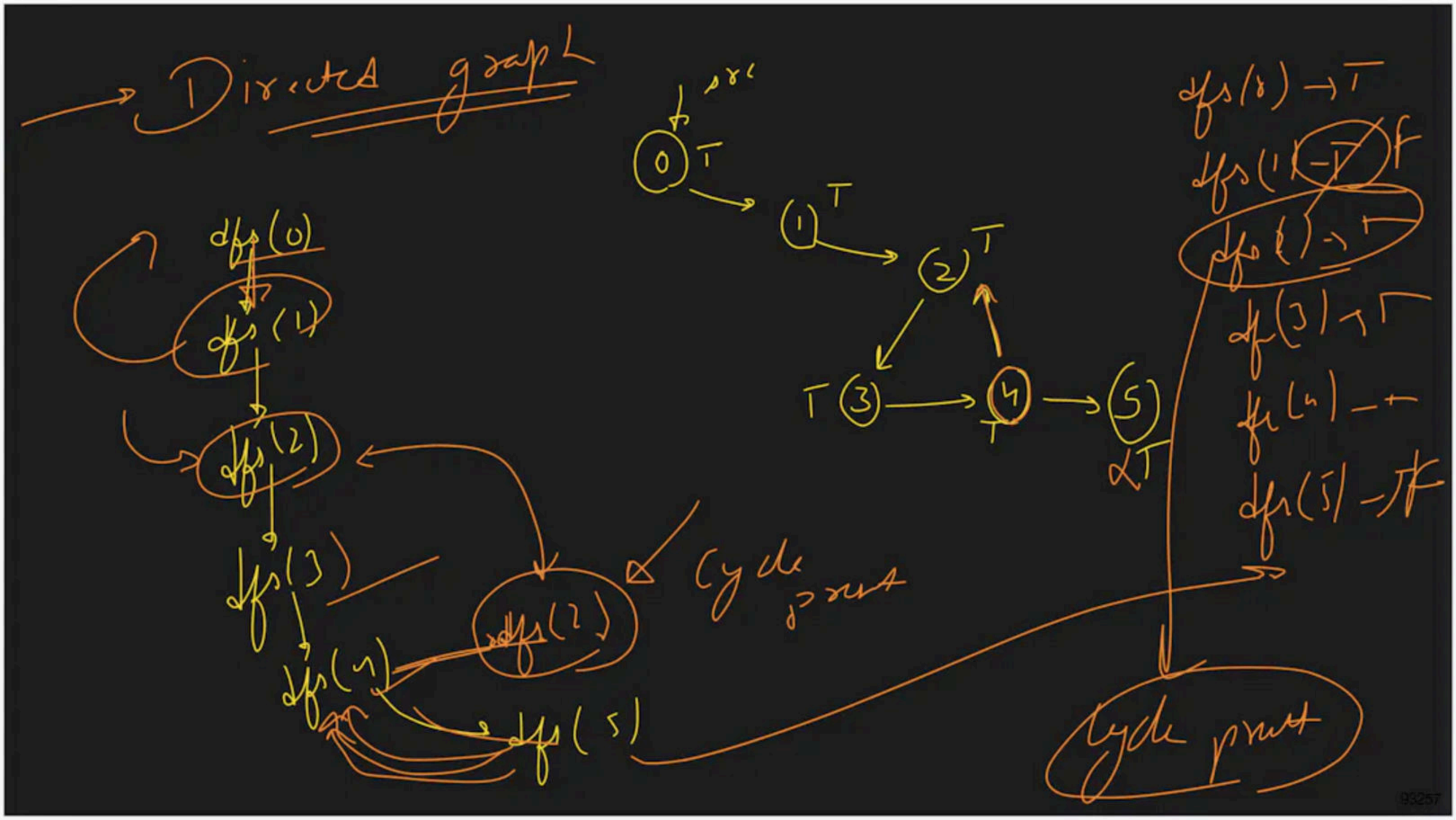
Nu. of Island

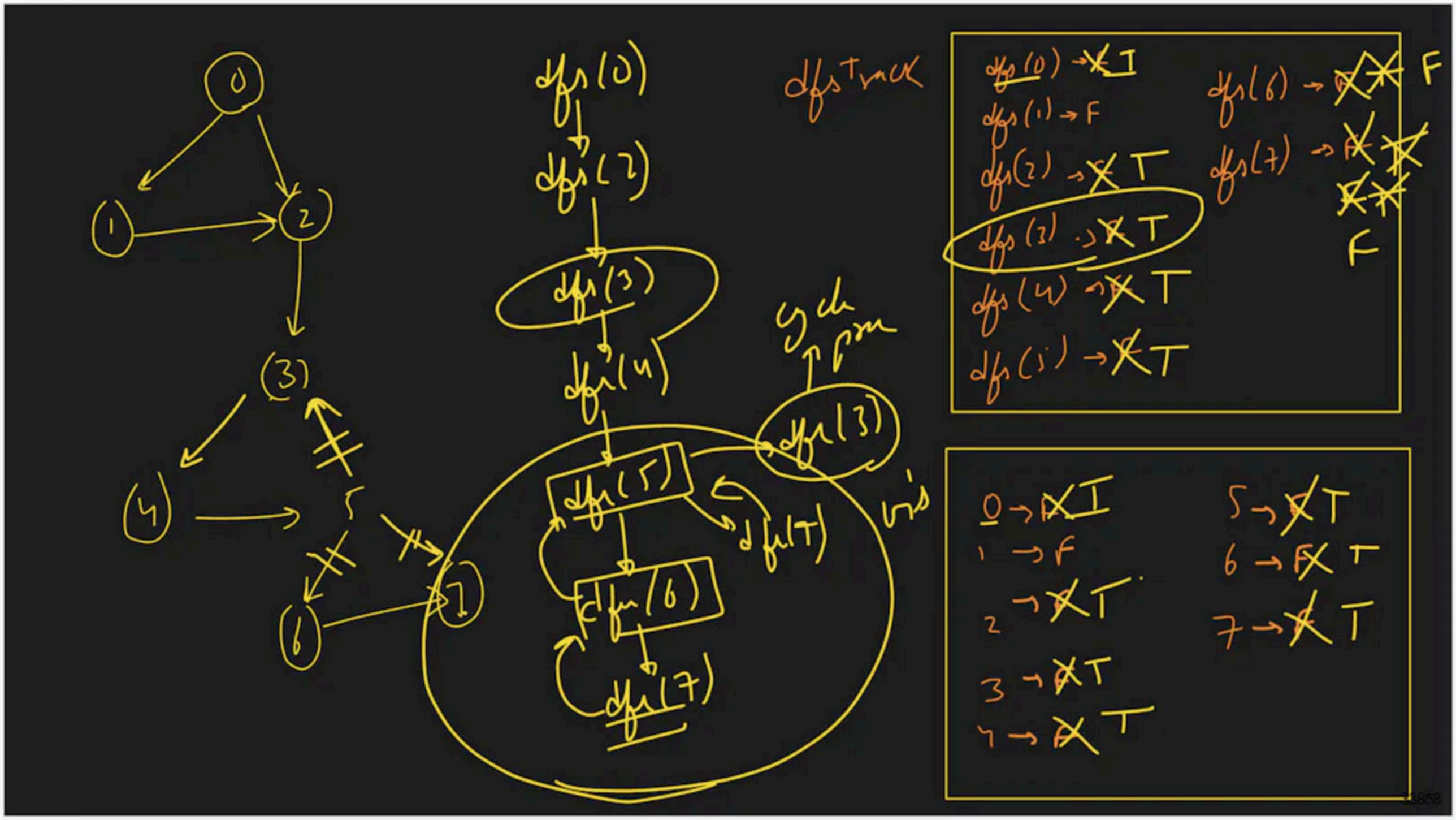
LC = 200

LL = 994 → Rotten On

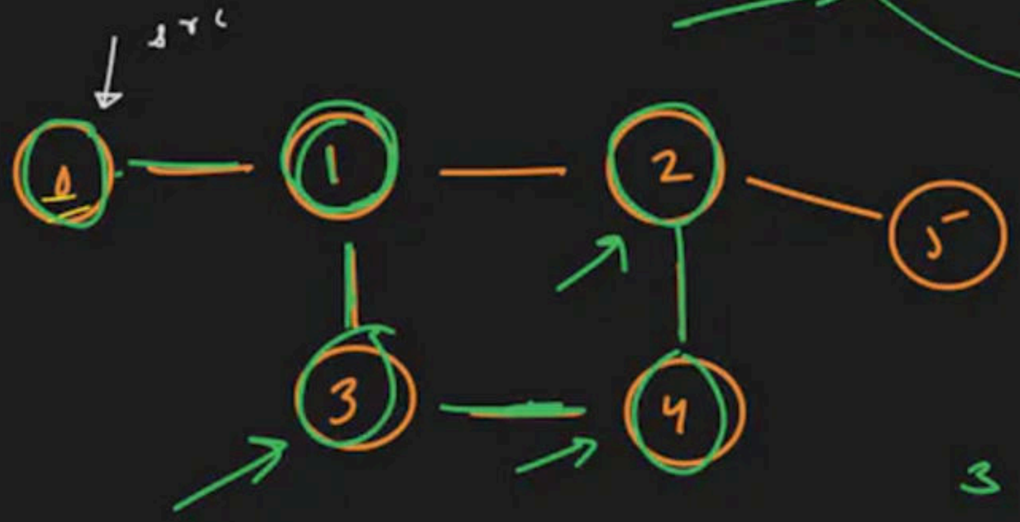
Cycle detection ::




Directed graph
src
0
T
1
T
2
T
3
T
4
T
5
dfs(0)
dfs(1)
dfs(2)
dfs(3)
dfs(4)
dfs(2)
dfs(5)
cycle present
dfs(0) → T
dfs(1) → T
dfs(2) → T
dfs(3) → T
dfs(4) → T
dfs(5) → T
cycle present
93257


0
1
2
3
4
5
6
7
dfs(0)
dfs(2)
dfs(3)
dfs(4)
dfs(5)
dfs(6)
dfs(7)
dfs(T)
dfs(3)
cycle from
vis
dfsTrack
dfs(0) → X I
dfs(1) → F
dfs(2) → X T
dfs(3) → X T
dfs(4) → X T
dfs(5) → X T
dfs(6) → X X F
dfs(7) → X X X F
0 → X I
1 → F
2 → X T
3 → X T
4 → X T
5 → X T
6 → F X T
7 → X T

Undirected :-



visited [child] == T
&&
child != parent [front]

cycle present

adj list

0 : {1}
1 : {0, 2, 3}
2 : {1, 4, 5}
3 : {1, 4}
4 : {2, 3}
5 : {2}

parent

0 : -1
1 : 0
2 : 1
3 : -1
4 : 2
5 : 2

Visited

0 → F T
1 → F T
2 → F T
3 → F T
4 → F T
5 → F T

Queue

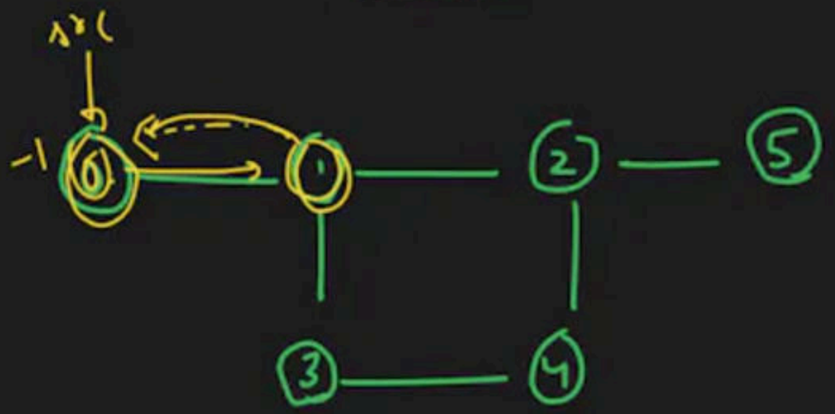
q.push(src)
vis[0] = T
par = -1

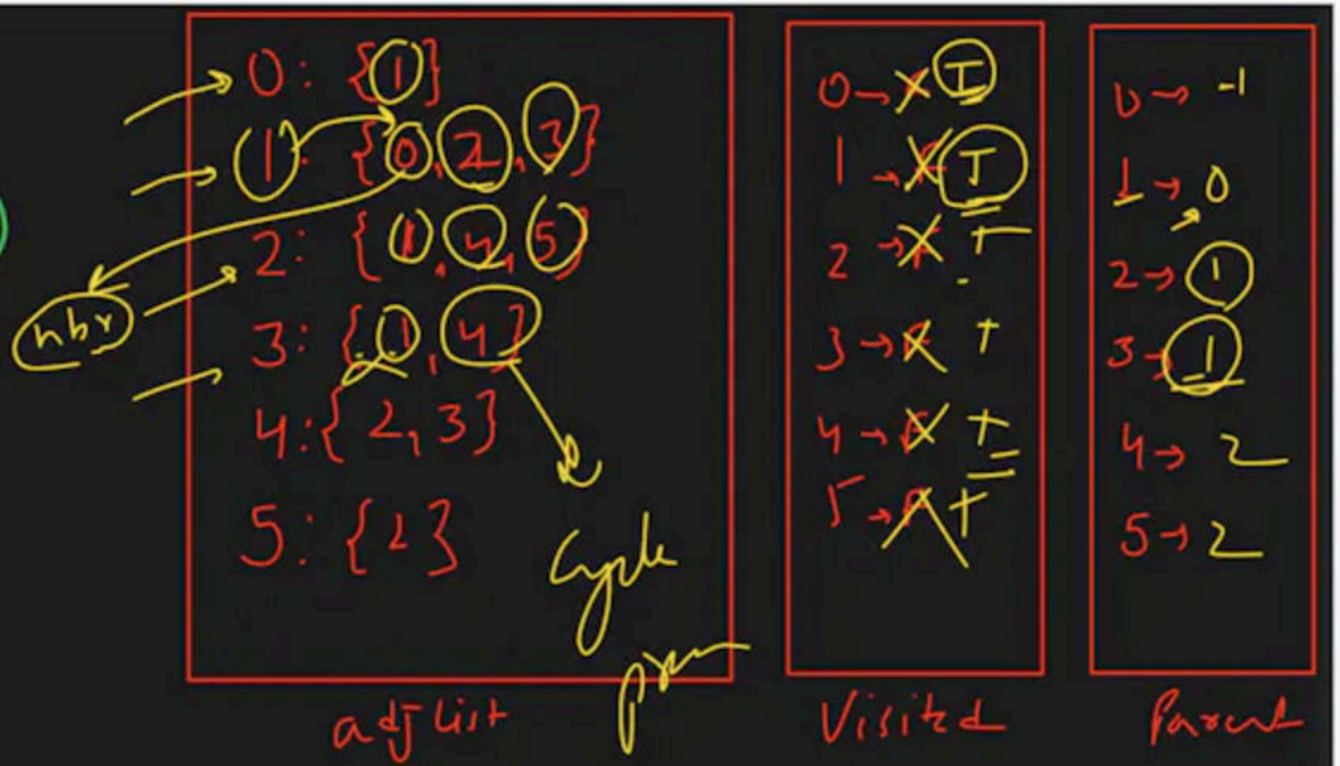
| 0 | 1 | 2 | 3 | 4 | 5 |

0 1 2 3

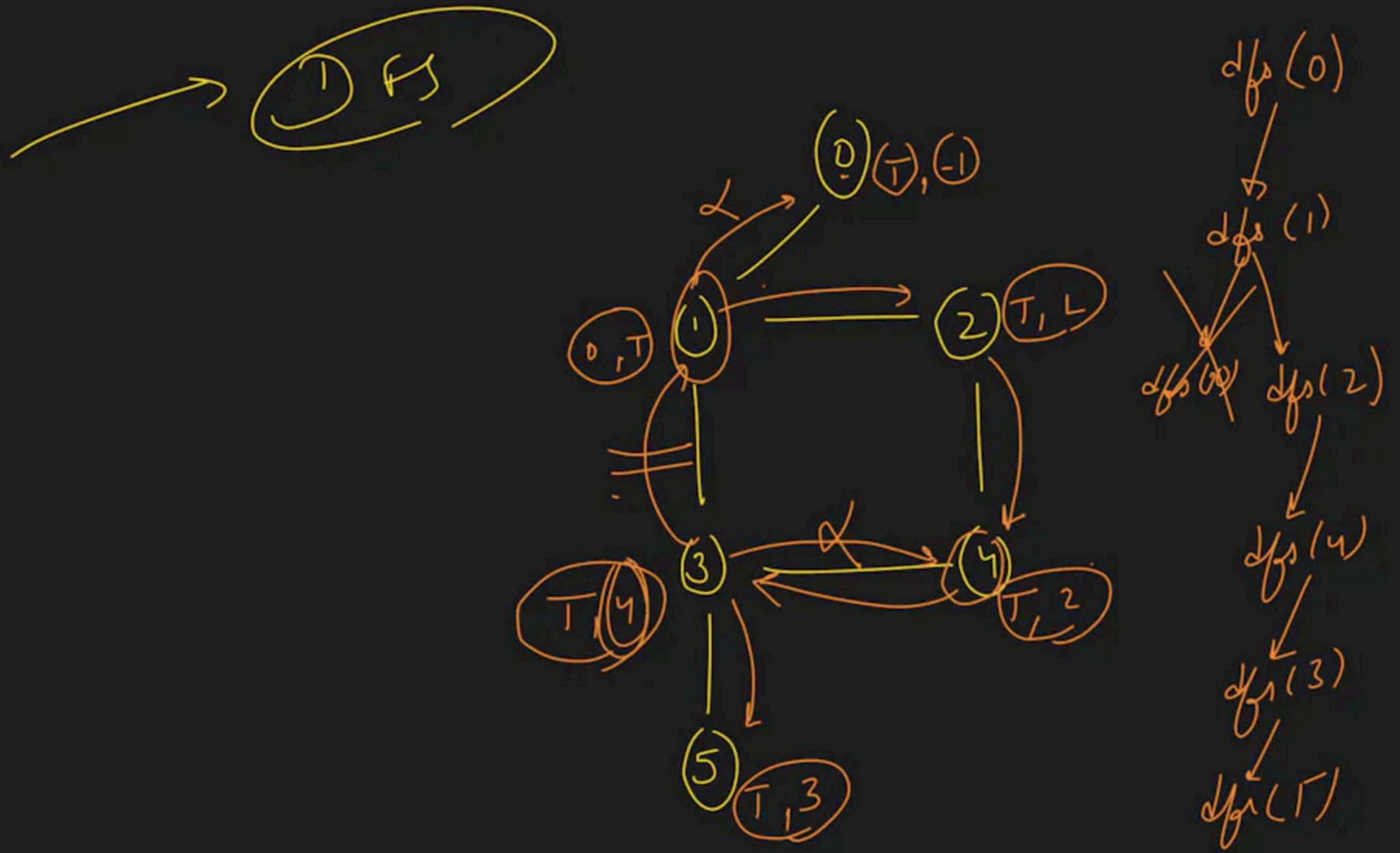
2 ← 1

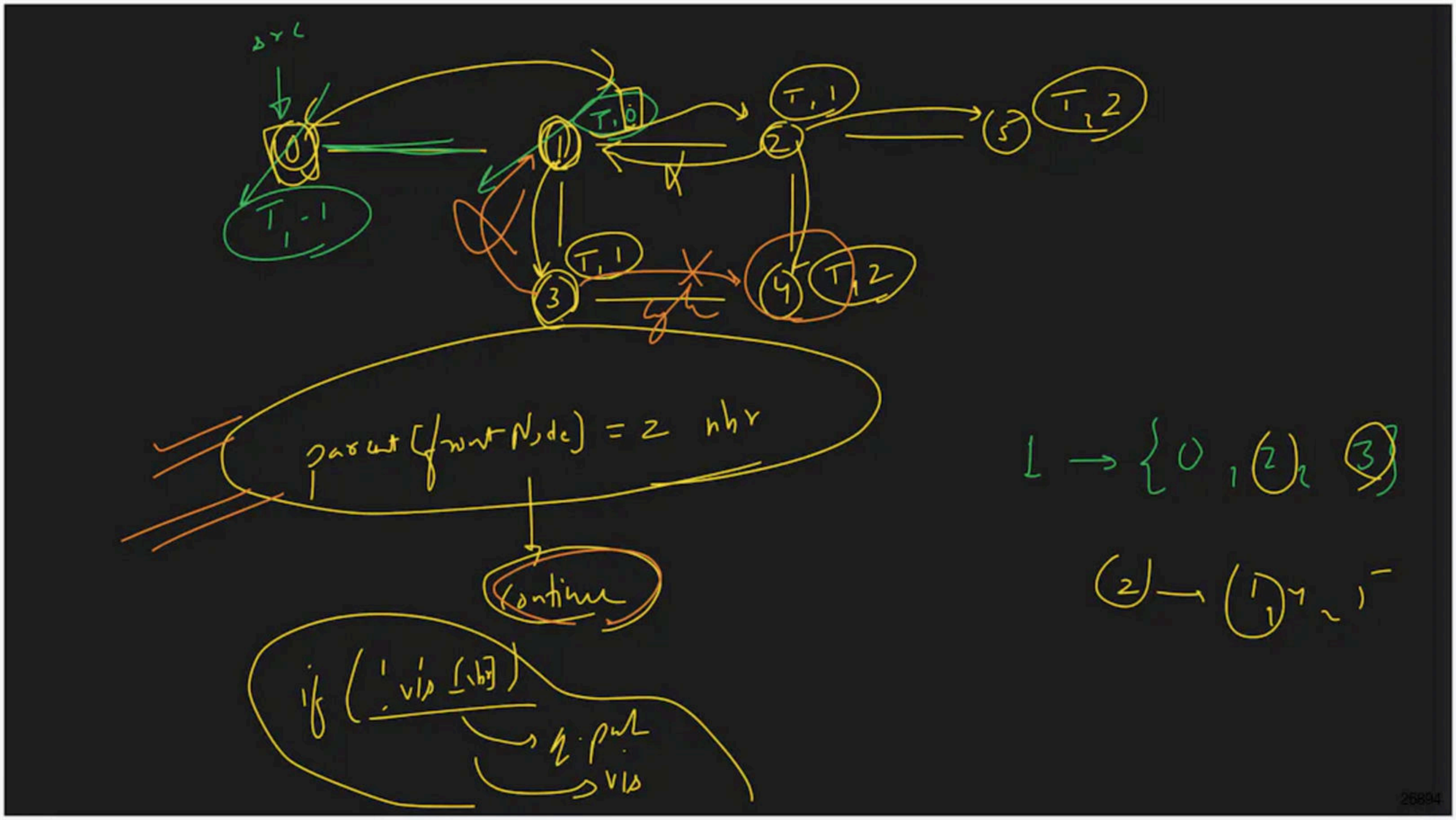
Cond^t → vis[nbr] == true
&&
nbr != parent[frontNode]

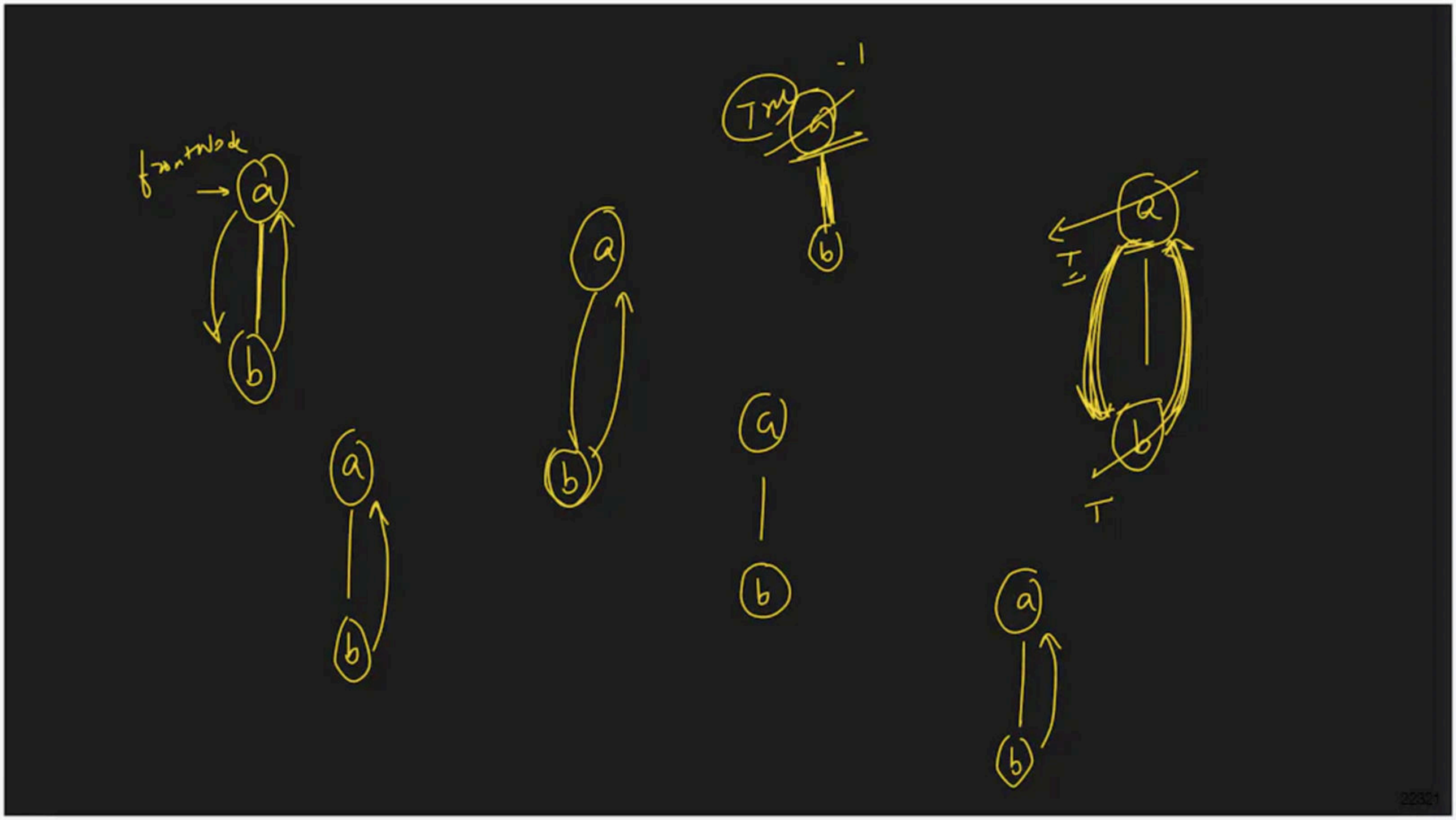


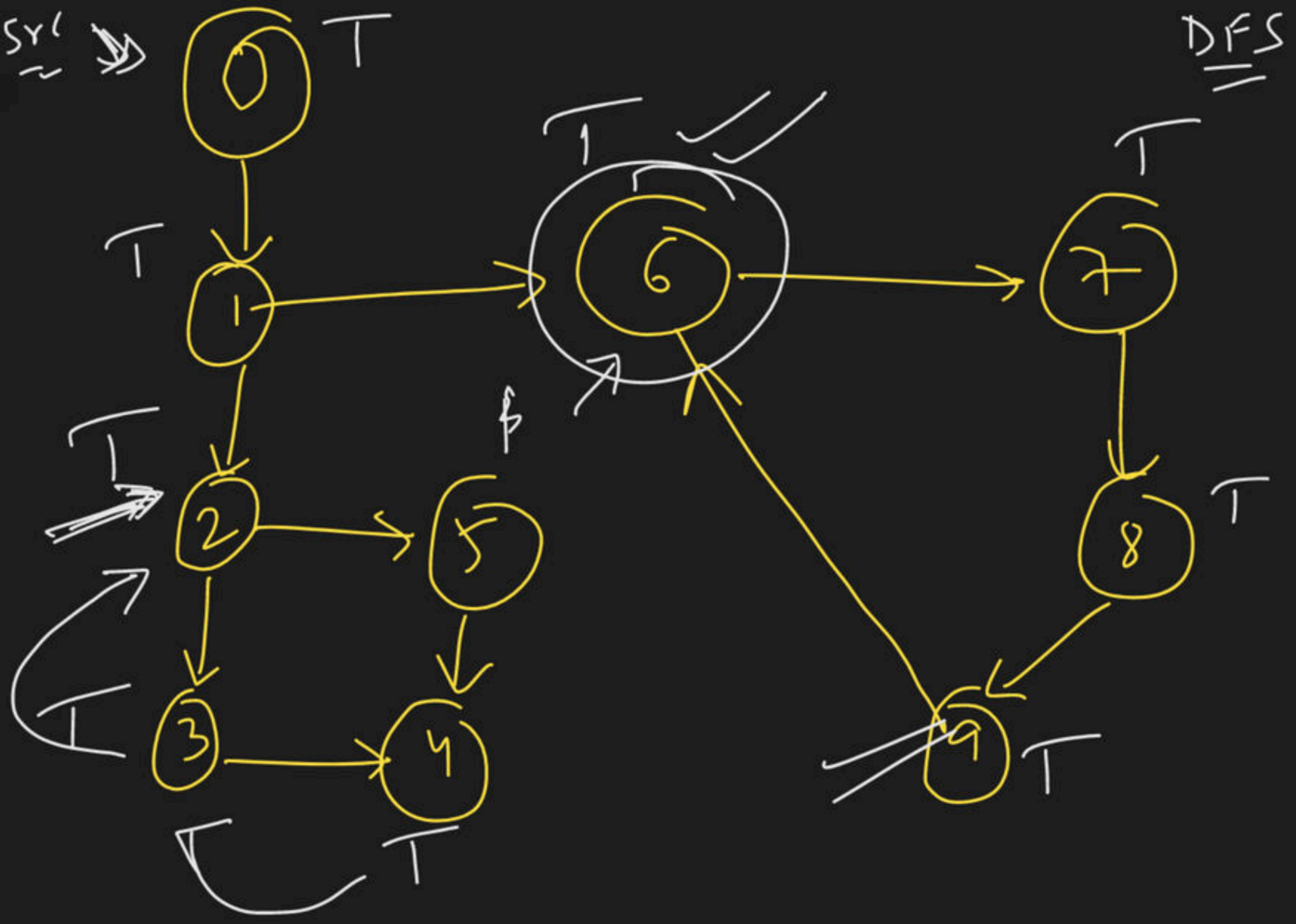


q.push(src)
vis[src] = T
parent[src] = -1


dfs(0)
dfs(1)
dfs(2)
dfs(4)
dfs(3)
dfs(5)


src
0
1
2
3
4
5
T,0
T,1
T,2
T,-1
parent[front Node] = 2 nhr
continue
if (! vis [nbr])
2 push
vis
1 → { 0 , 2 , 3 }
2 → (1)^{n-1}


a
b
a
b
a
b
a
b
a
b

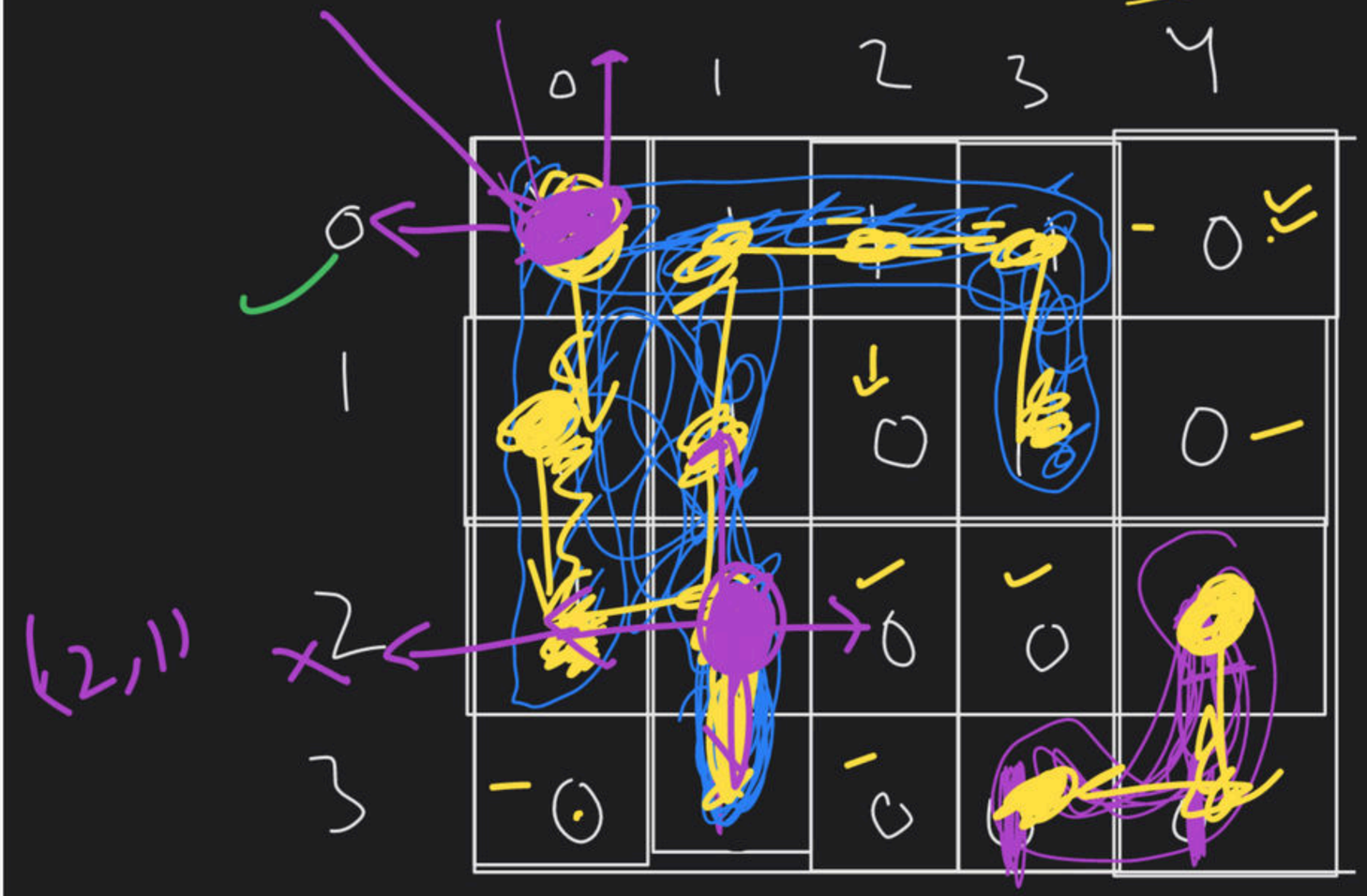


Visited Map

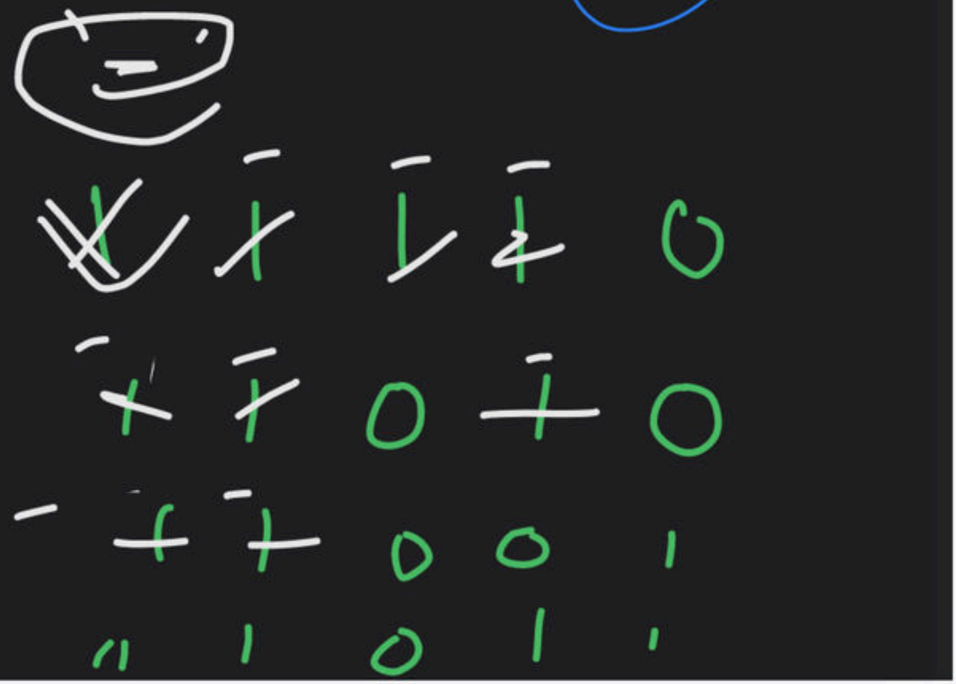
dfsTrue

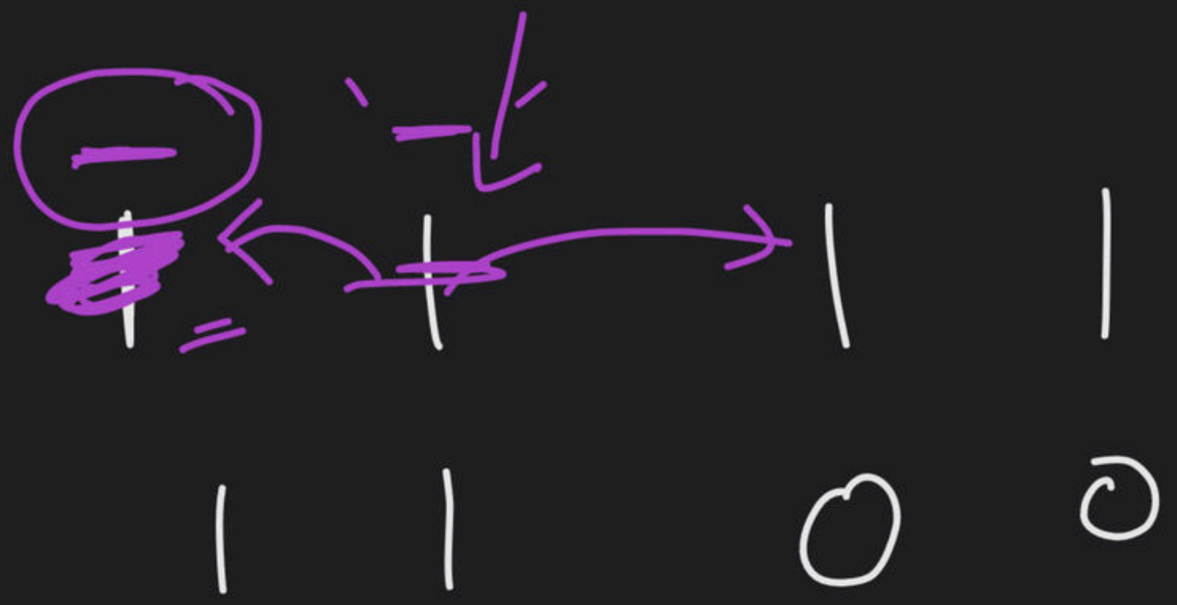
0 T	0 T
1 T	1 T
2 T	2 T
3 T	3
4 T	4
5	5
6	6 T
7	7
8	8
9	9

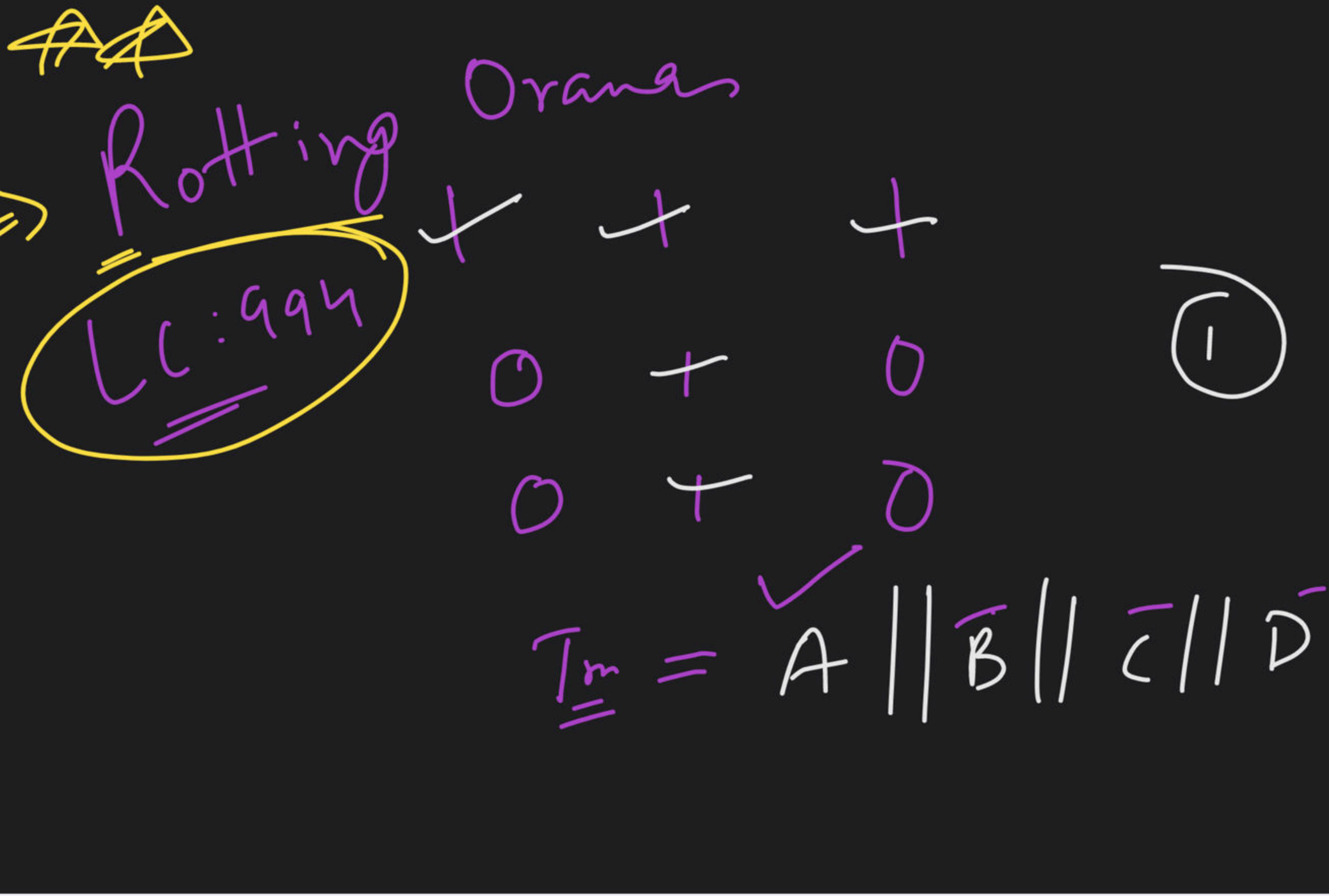
LL = 200 - No. of islands



① Out of range
② Water Rules






Rotting Oranges
LC: 994
1
$Tm = A \,||\, \bar{B} \,||\, \bar{C} \,||\, \bar{D}$