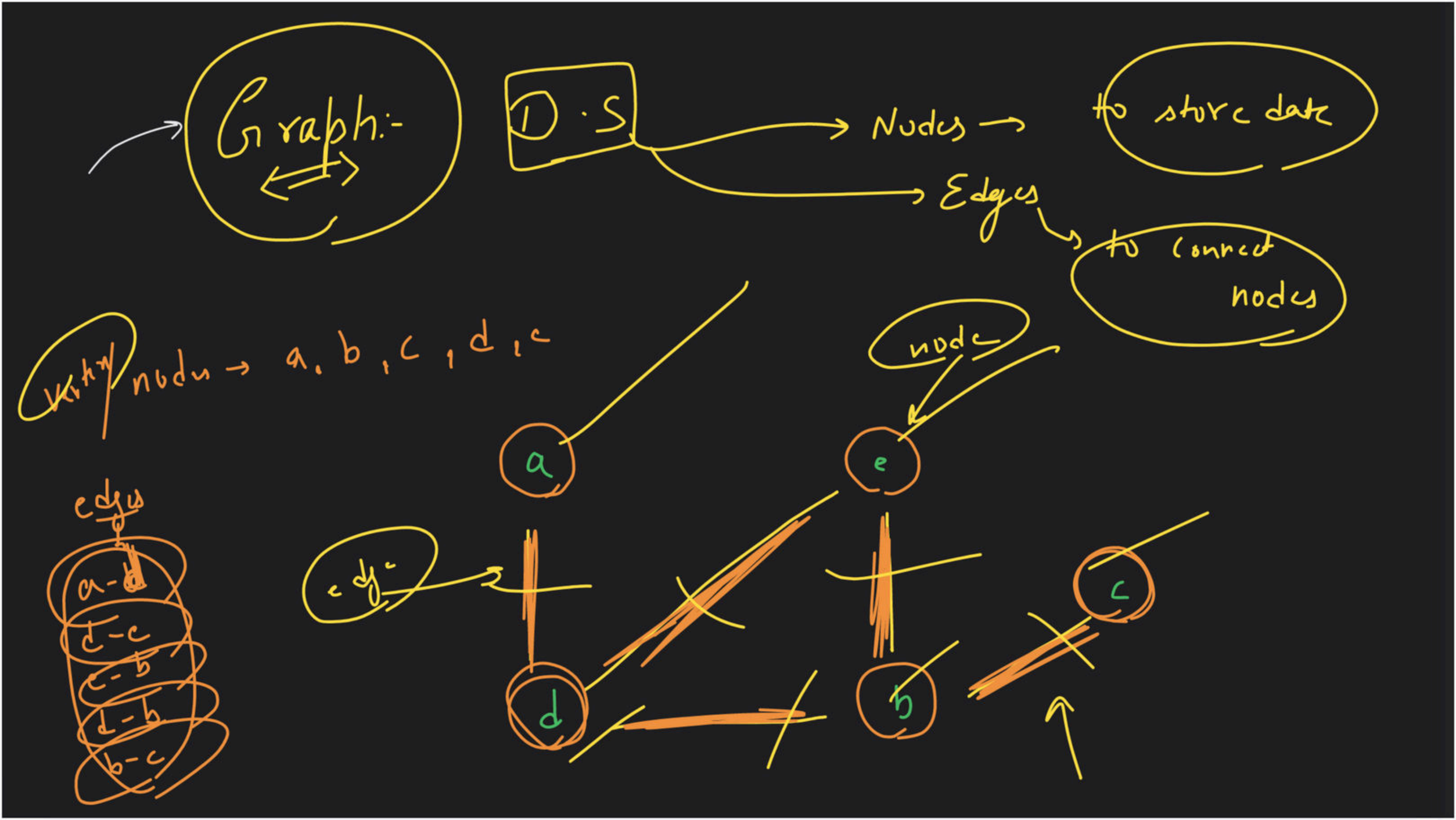
Graphs Class - 1

Special class

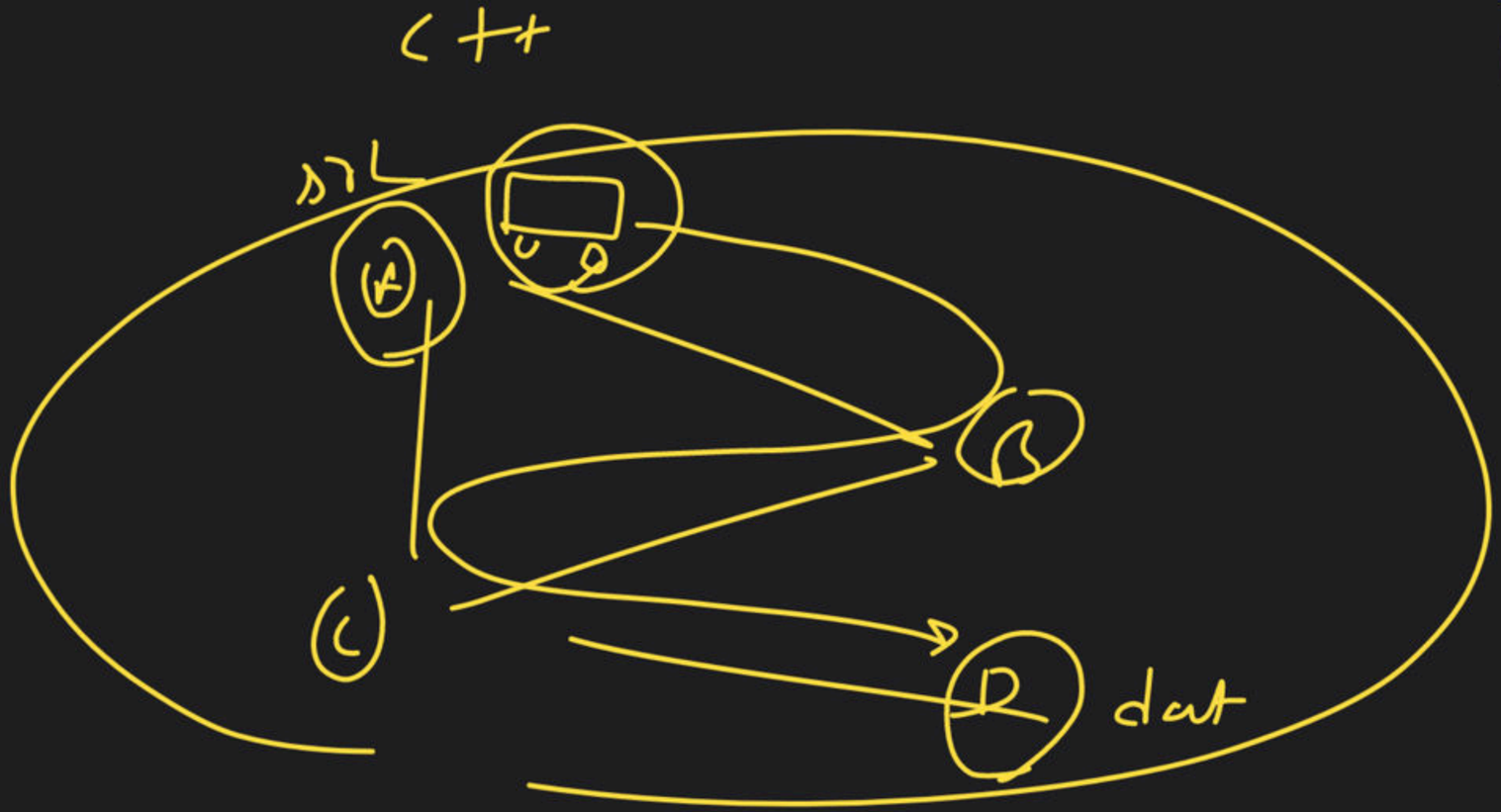
Love Babbar • June 17, 2023


Graph:-
D.S
Nodes →
to store data
Edges
to connect nodes
Vertex
nodes → a, b, c, d, e
node
a
e
edges
edge
a-d
d-c
e-b
d-b
b-c
d
b
c

2 Week

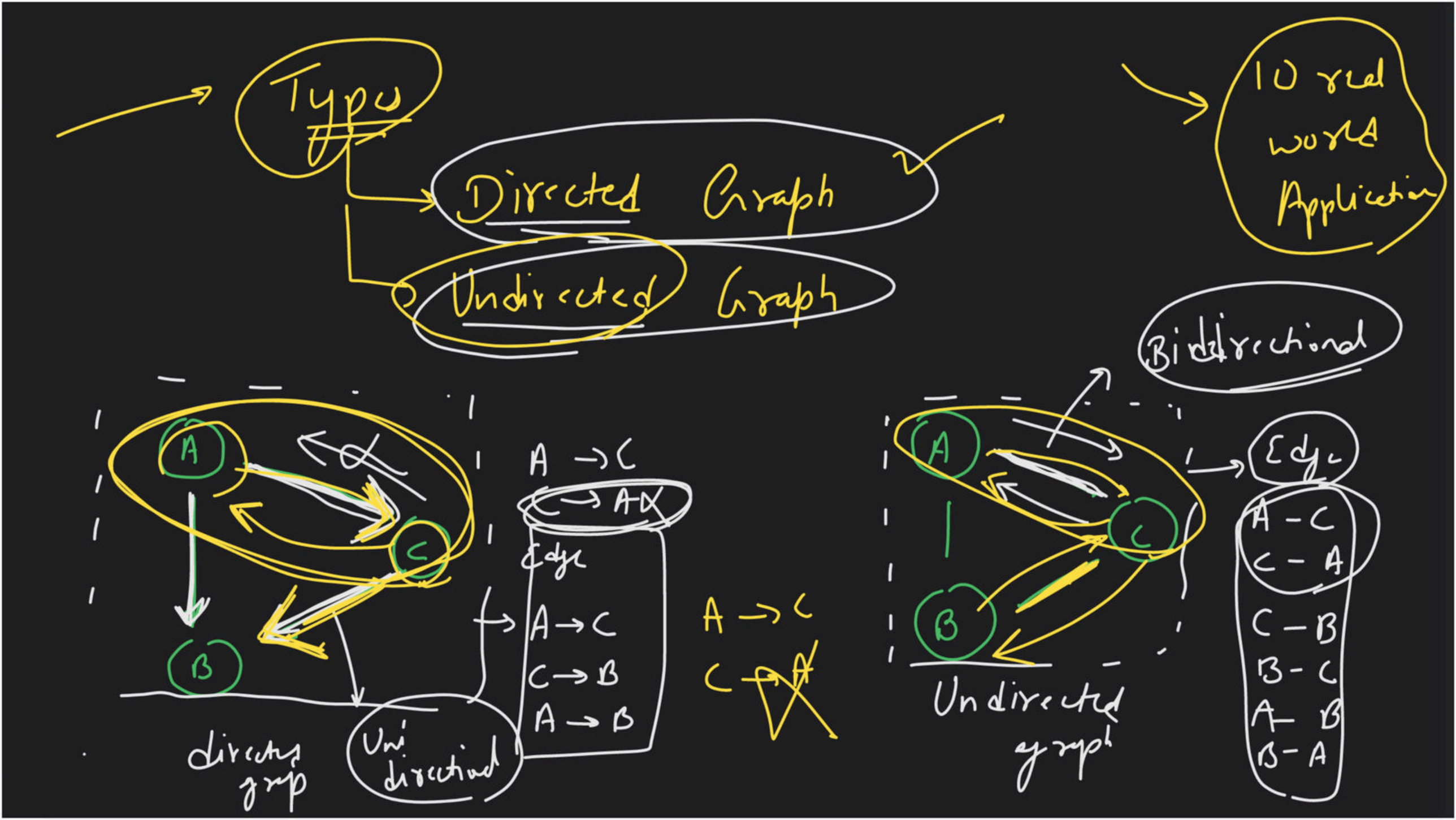
Resume

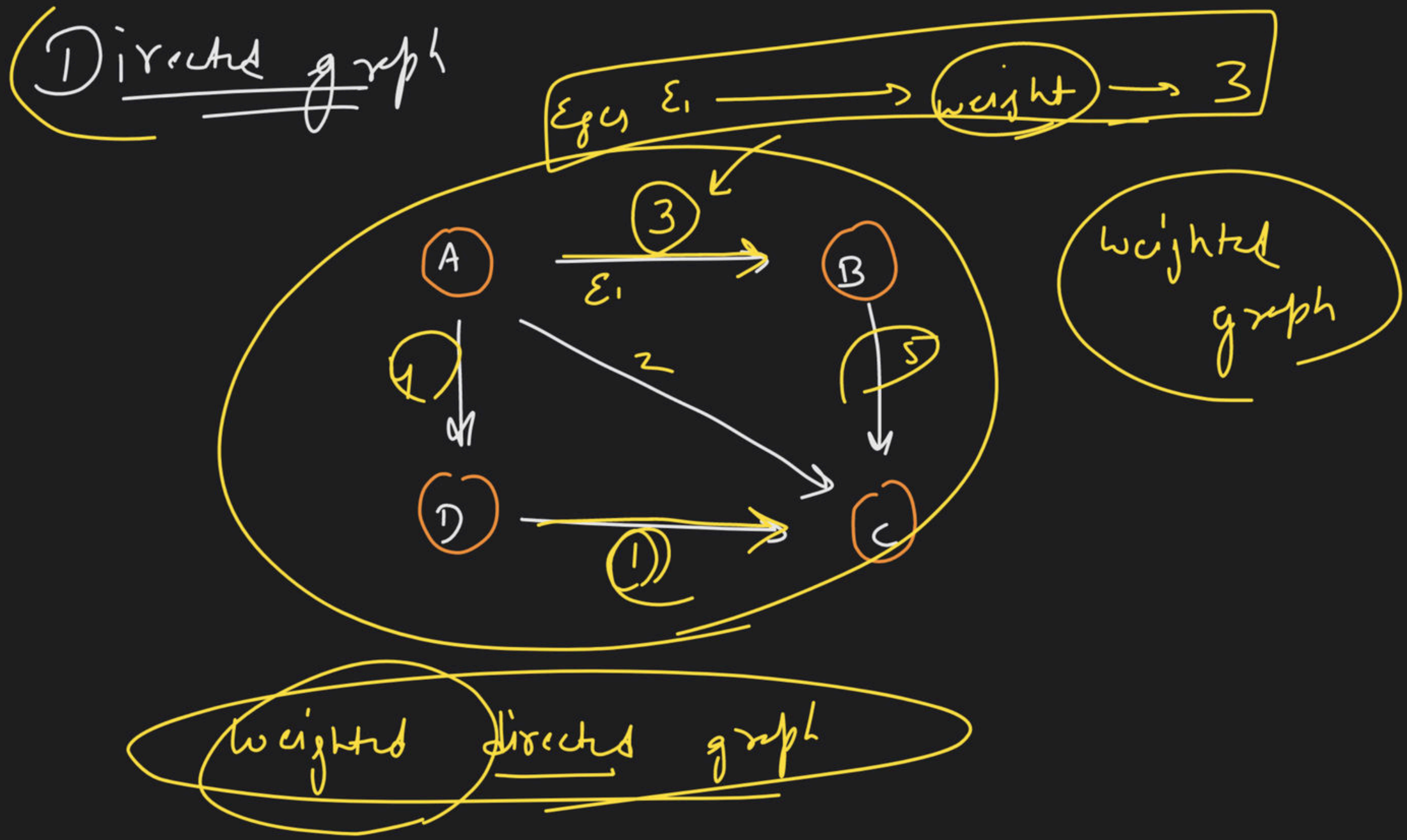
Bus System



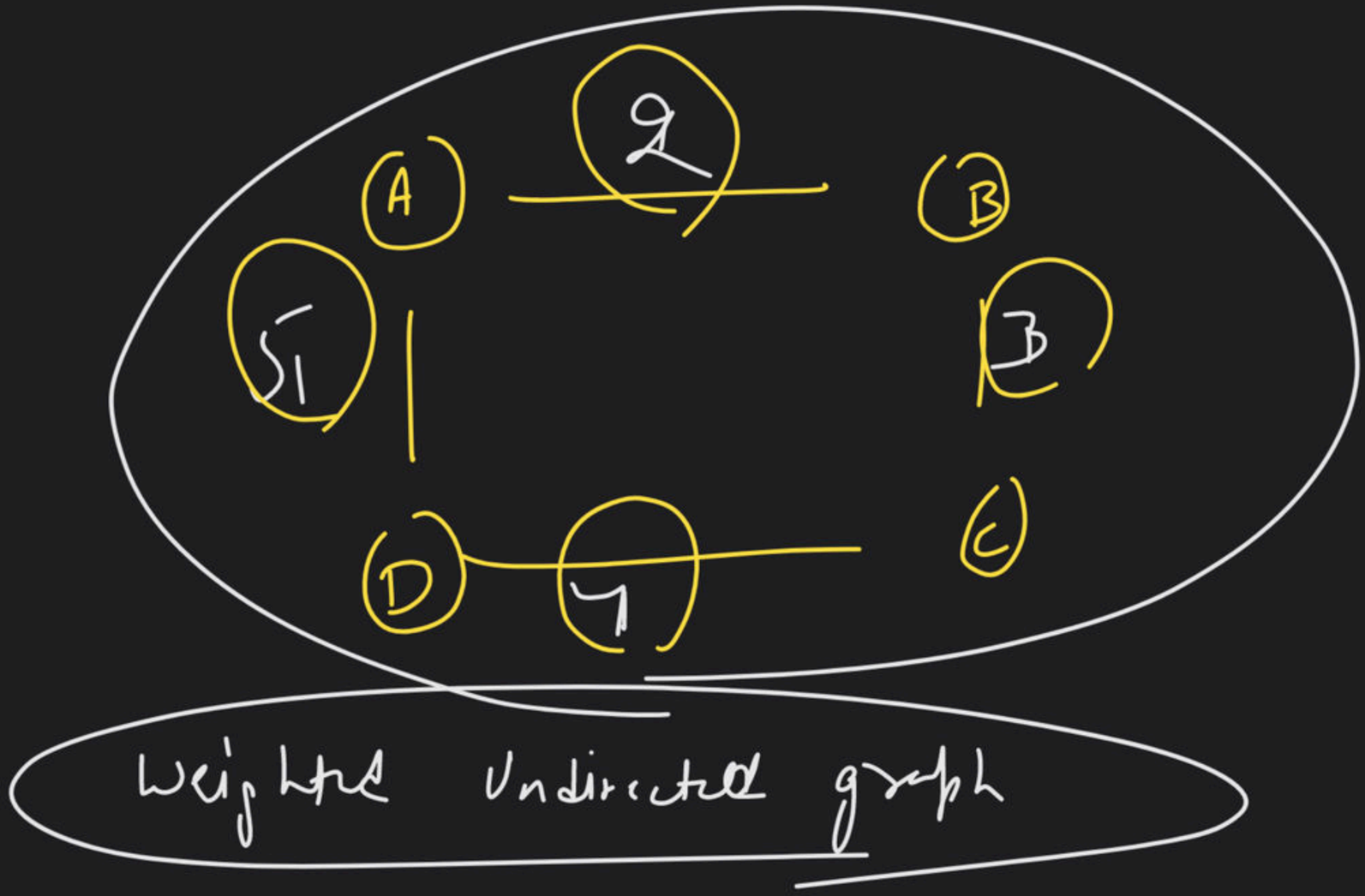
shortest path

↳ Dijkstra's M


Types
Directed Graph
Undirected Graph
10 real world Application
Bidirectional
A
B
C
A → C
C → B
A → B
Edge
directed graph
Uni directional
A → C
C → A
Undirected graph
Edge
A - C
C - A
C - B
B - C
A - B
B - A

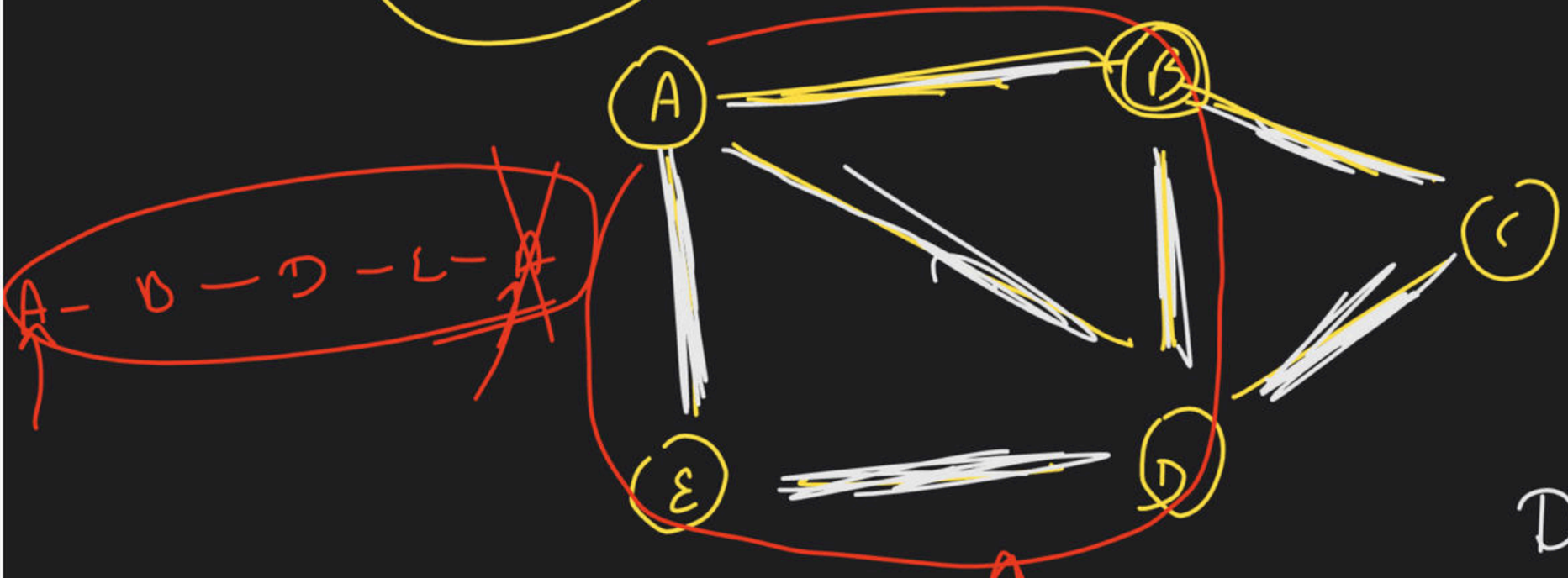

Directed graph
Eg: E_1 → weight → 3
3
A
E_1
B
4
2
5
D
1
C
weighted graph
weighted directed graph

Undirected graph



Degree

Degree(D) = 4



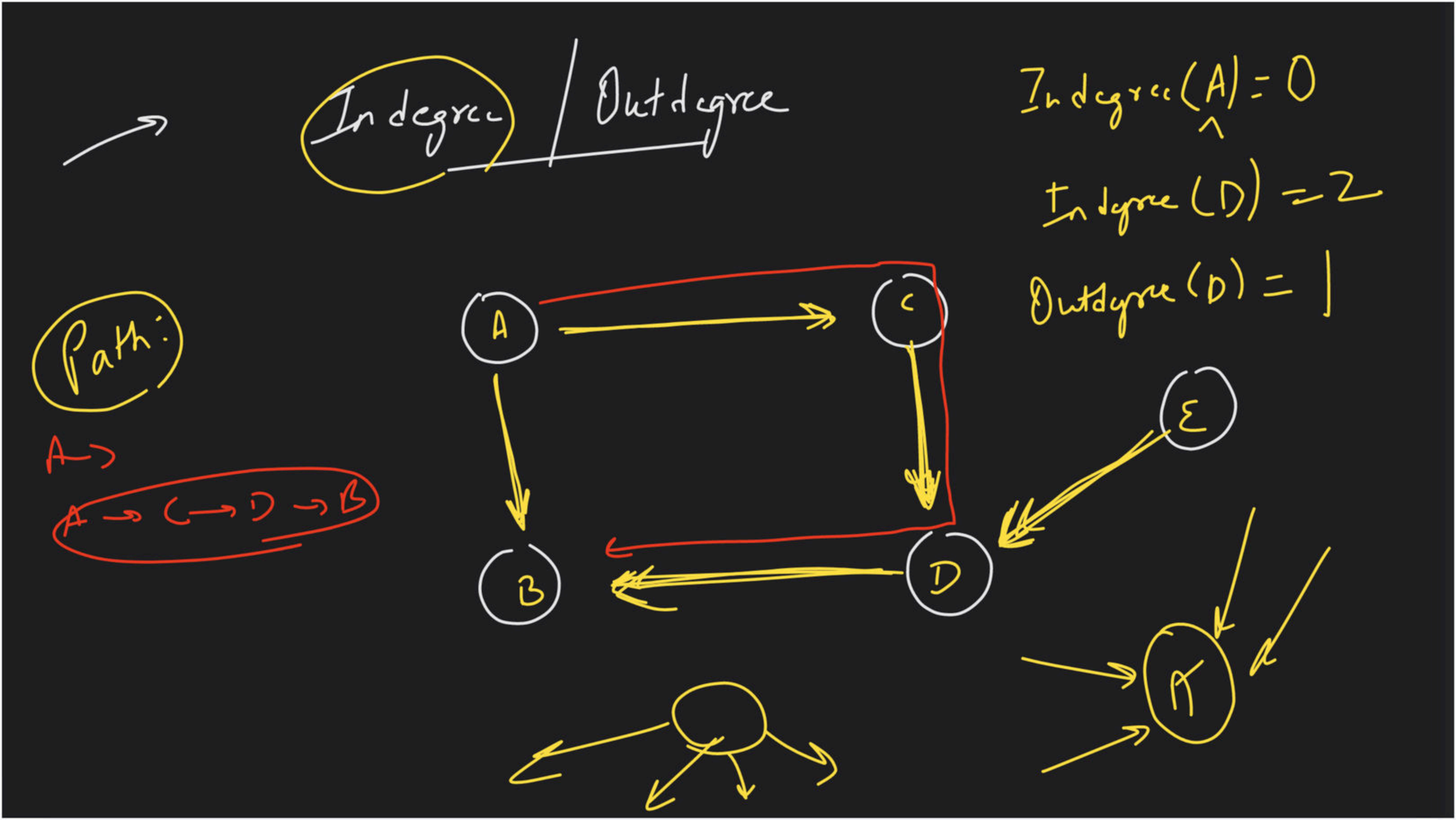
A - B - D - E - A

Degree(A) = 3

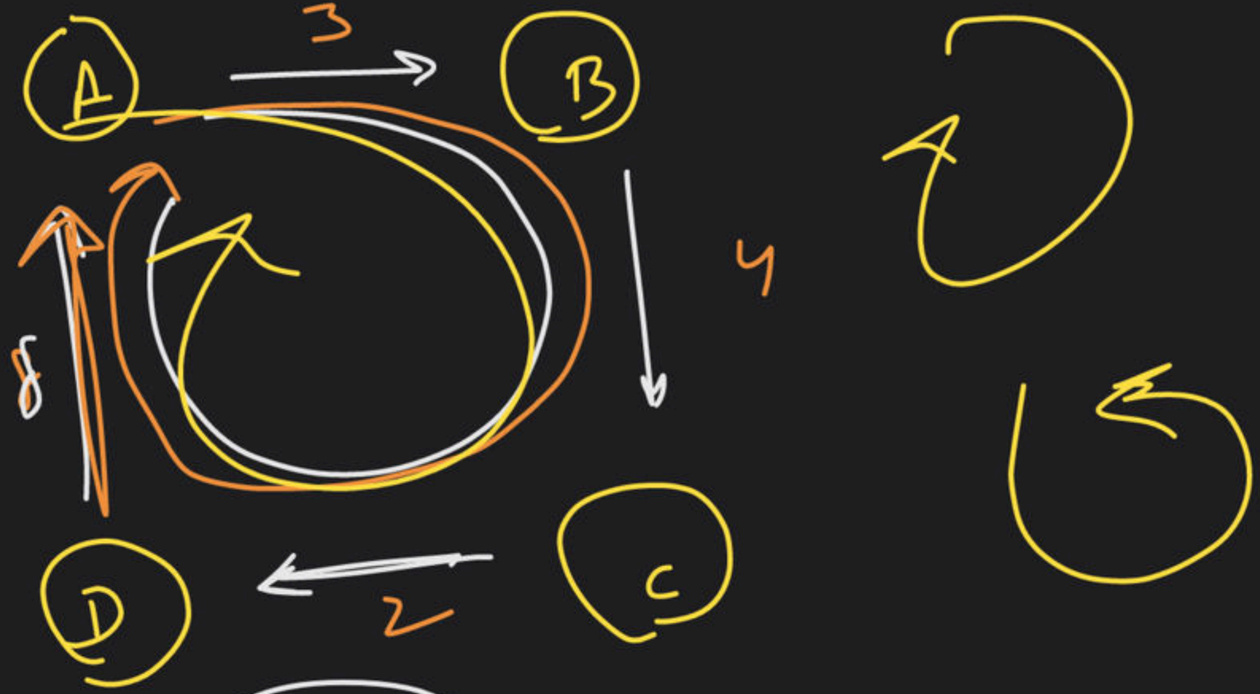
Degree(C) = 2

Degree(B) = 3

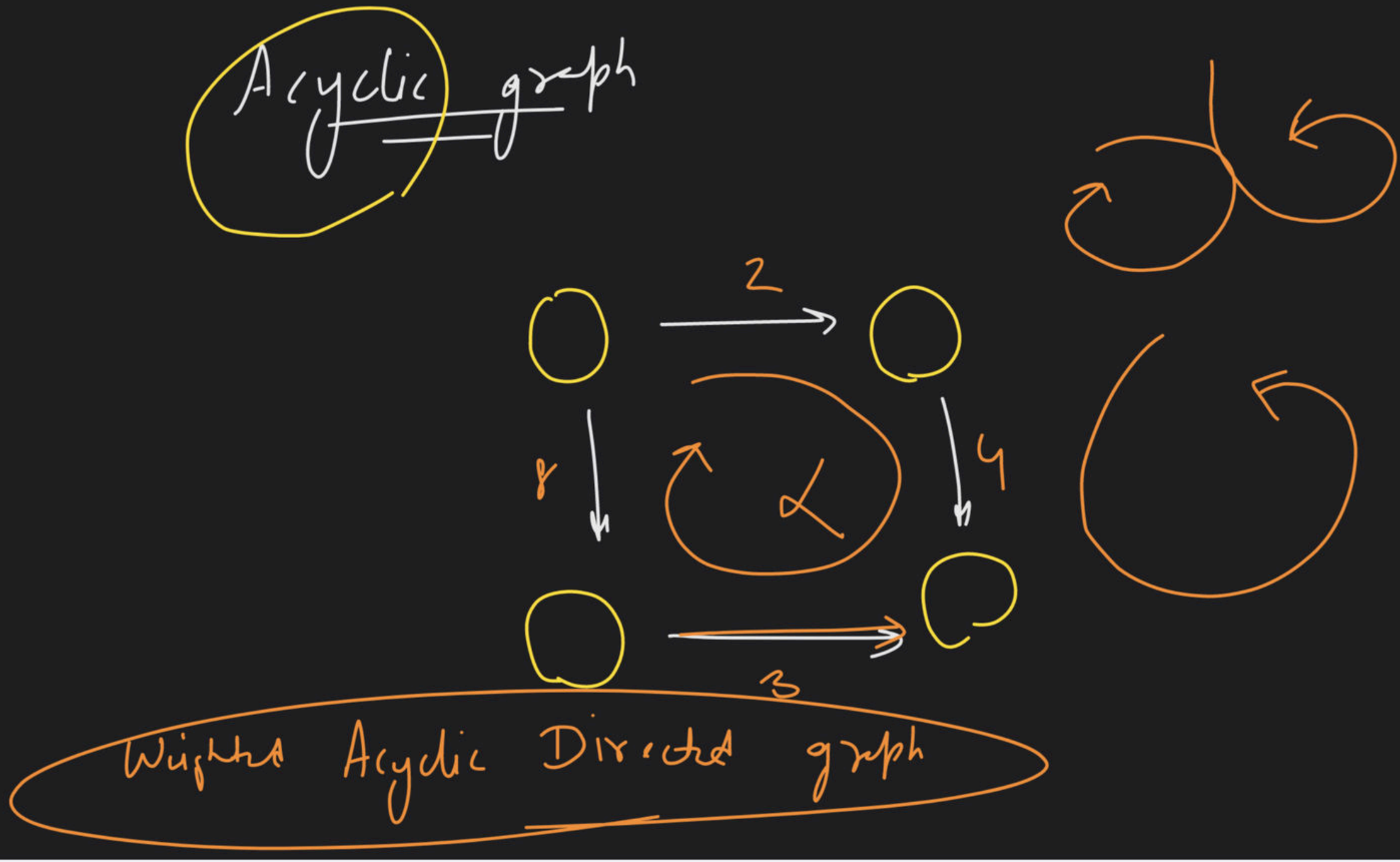
Degree(E) = 2

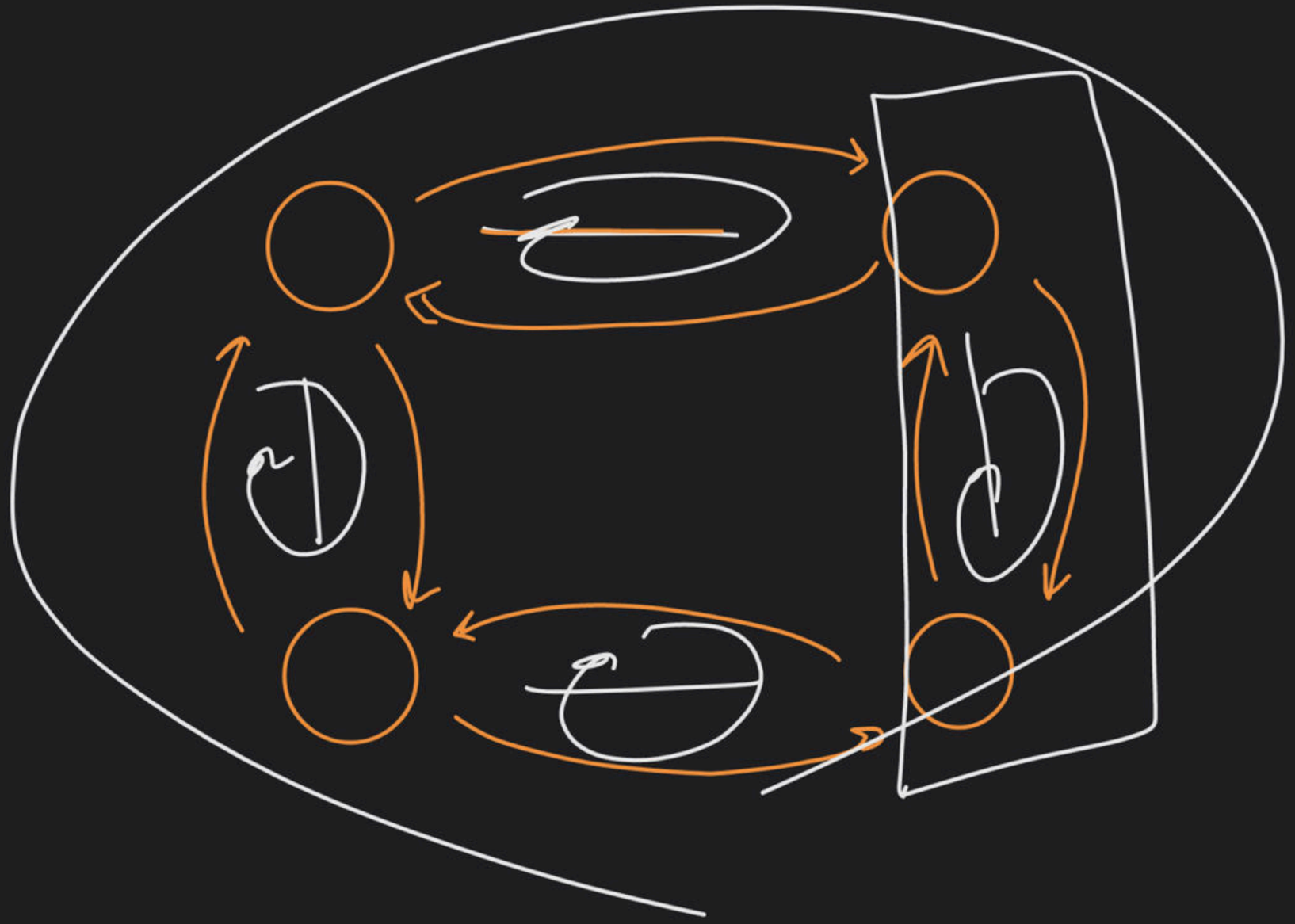

Indegree / Outdegree
Indegree(A) = 0
Indegree(D) = 2
Outdegree(D) = 1
Path:
A->
A → C → D → B
A
C
B
D
E
F

Cyclic graph:-

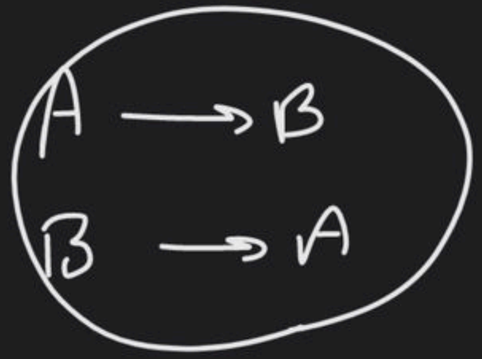


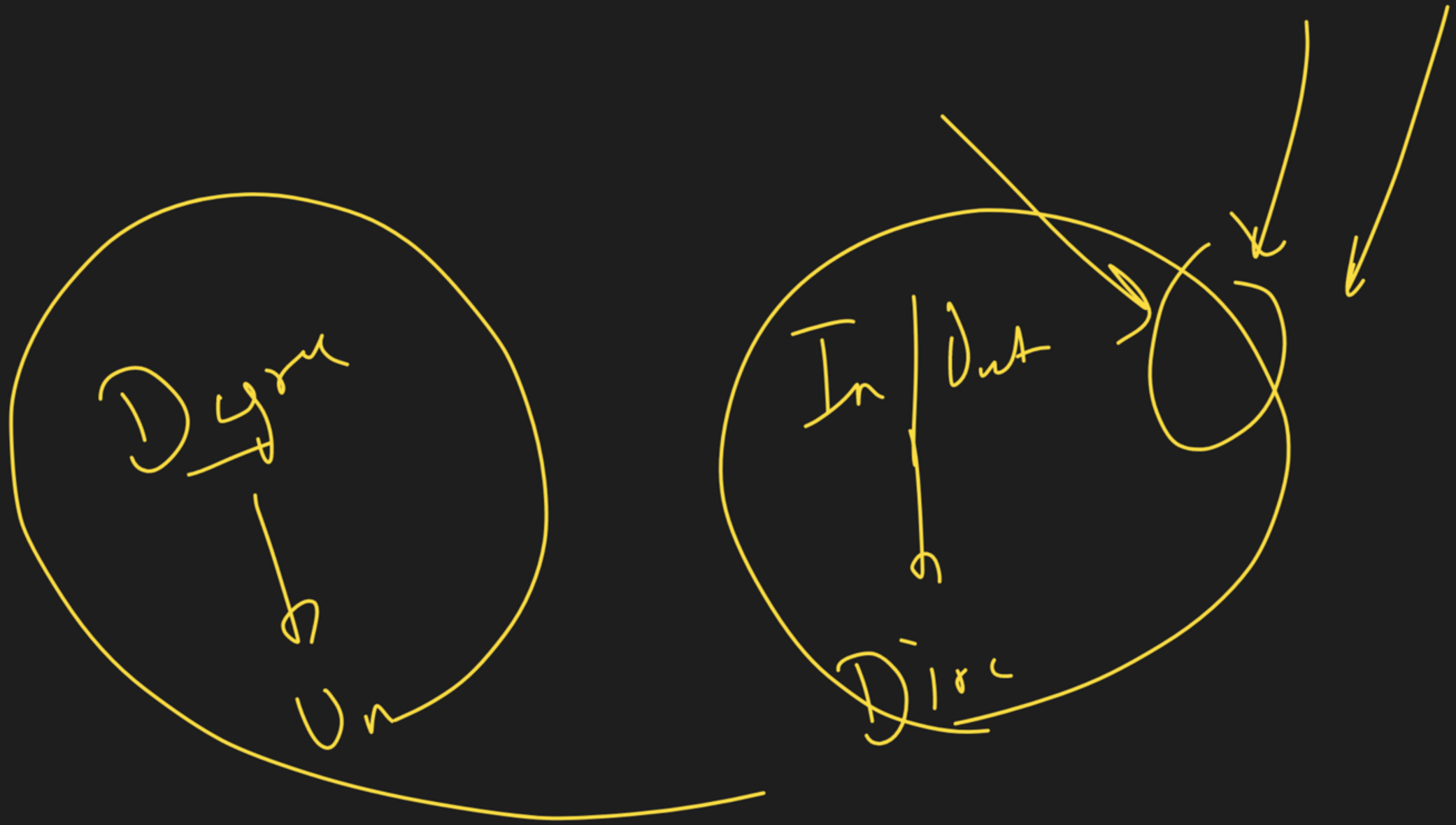
Weighted Cyclic Directed graph


Acyclic graph
2
8
4
3
Weighted Acyclic Directed graph

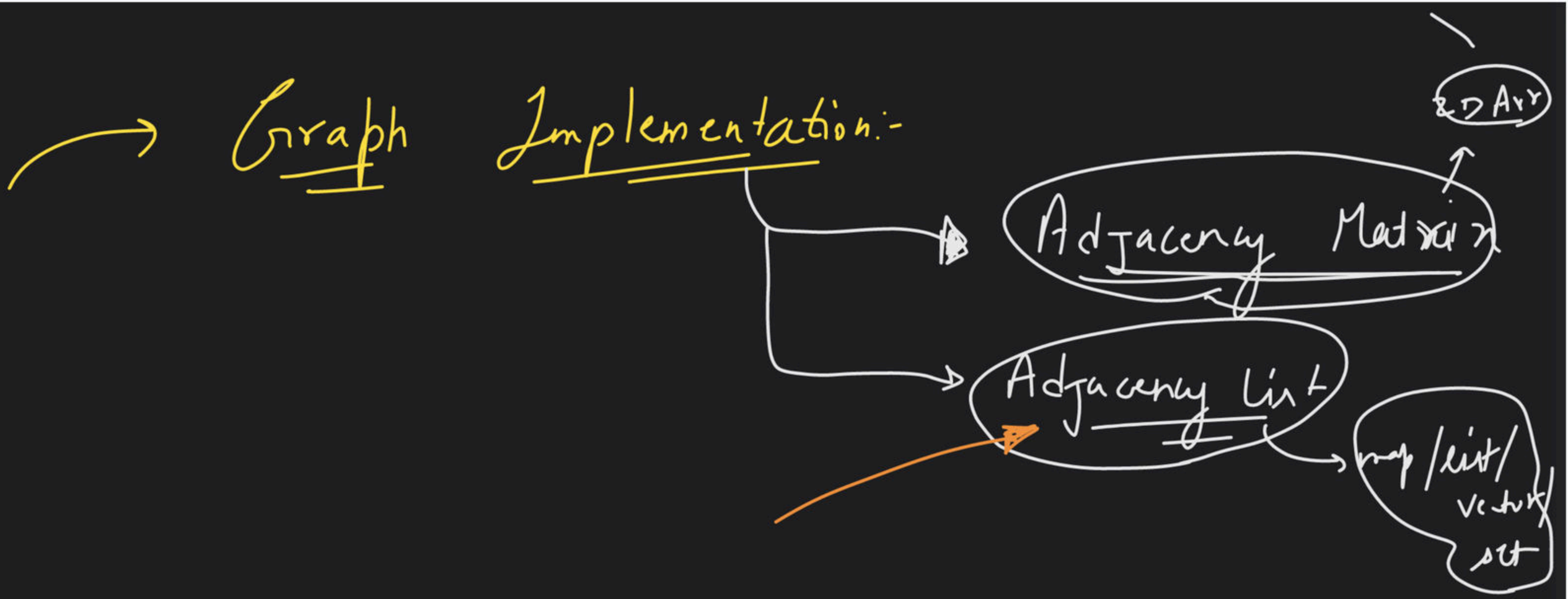



A
B


A → B
B → A


Dynamic
Un
In
Out
Direc

Graph Implementation:-



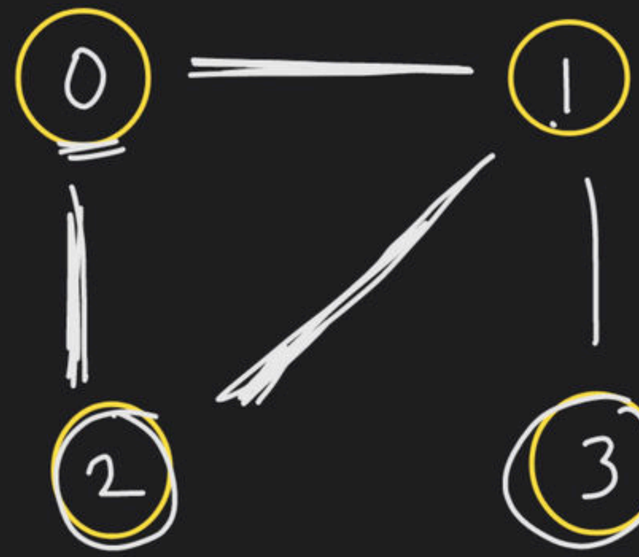
Adjacency matrix

$n = 4$

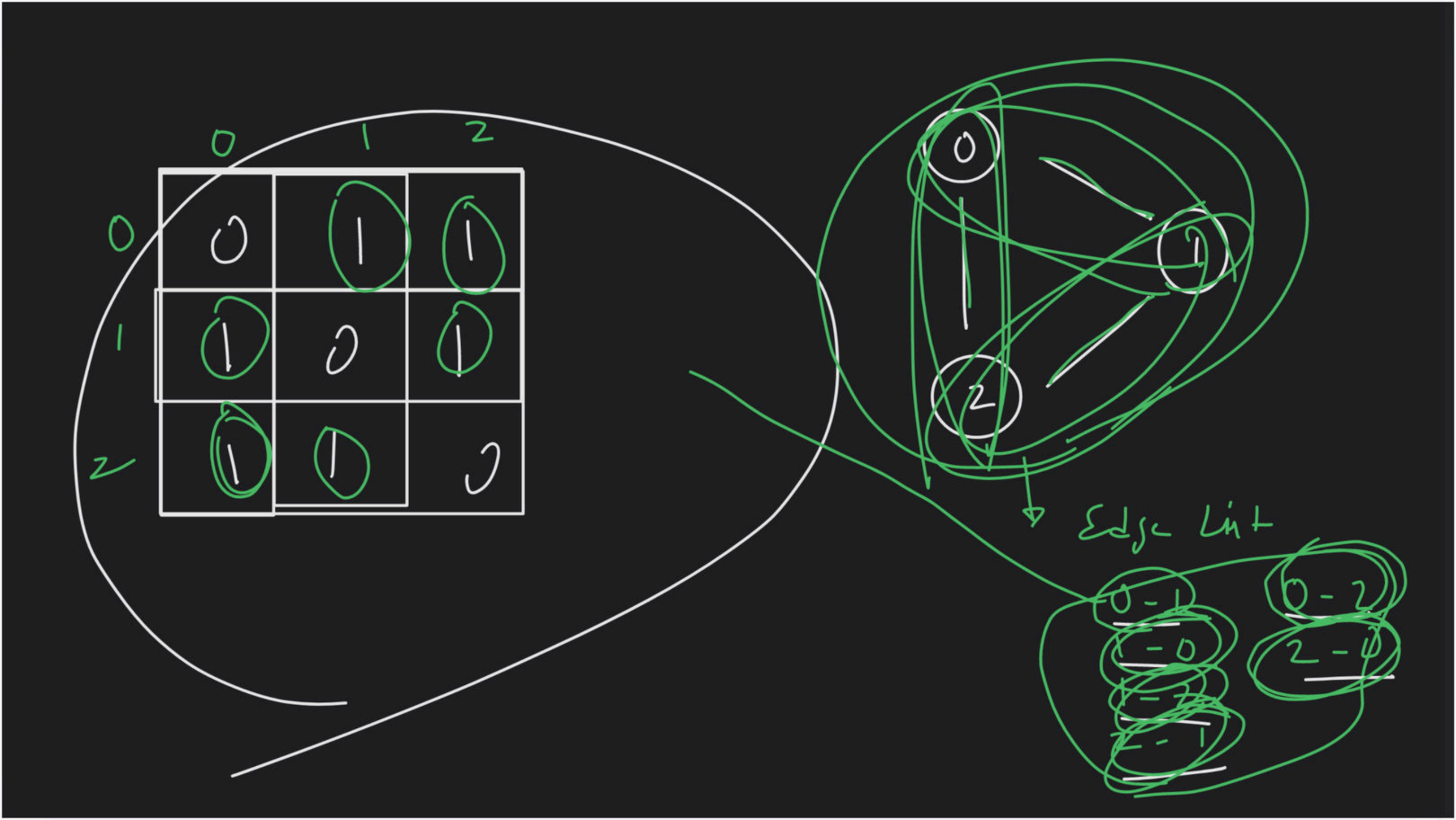
n = no of nodes

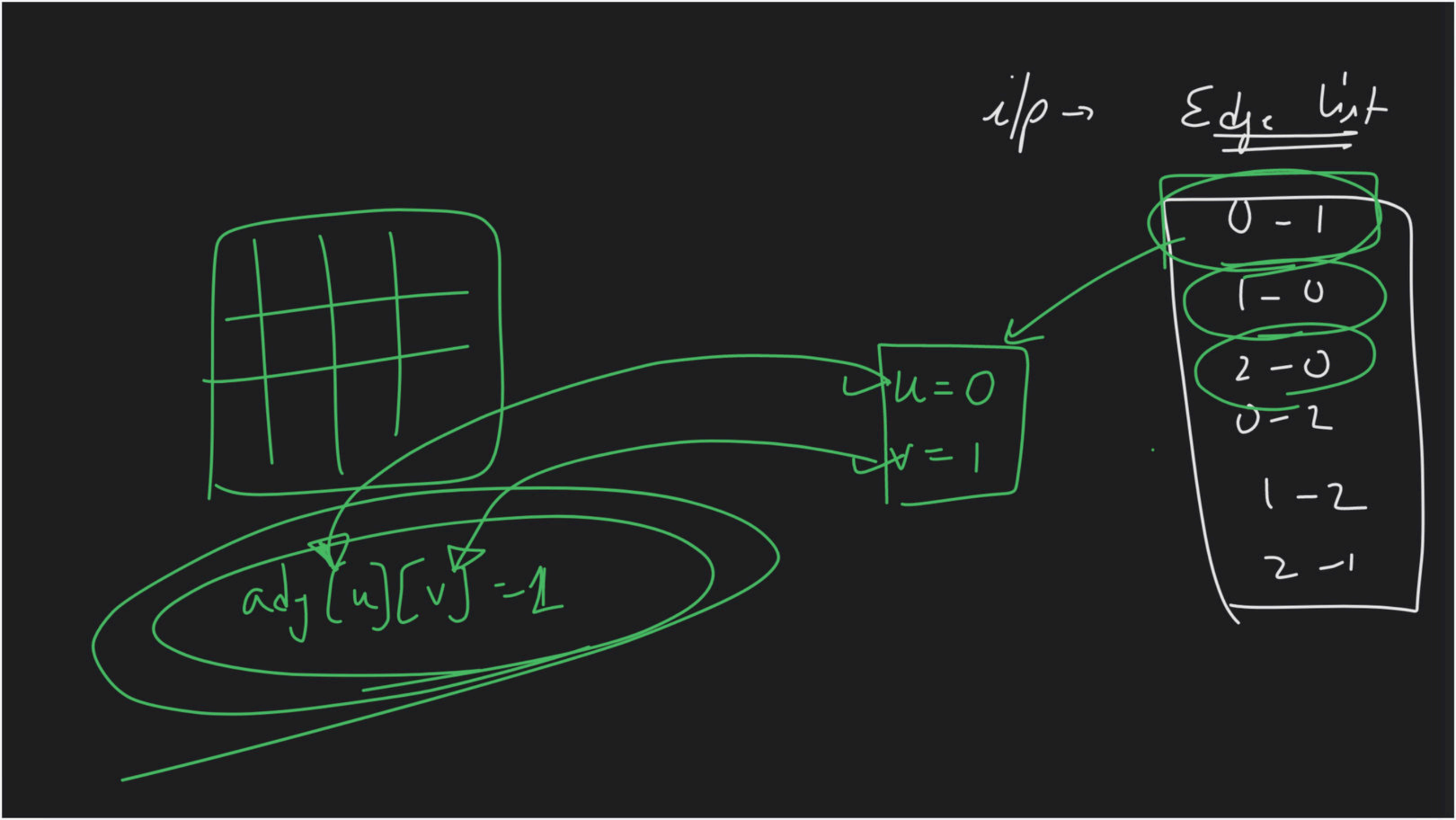
	0	1	2	3
0	0	1	1	0
1	1	0	1	1
2	1	1	0	0
3	0	1	0	0

graph

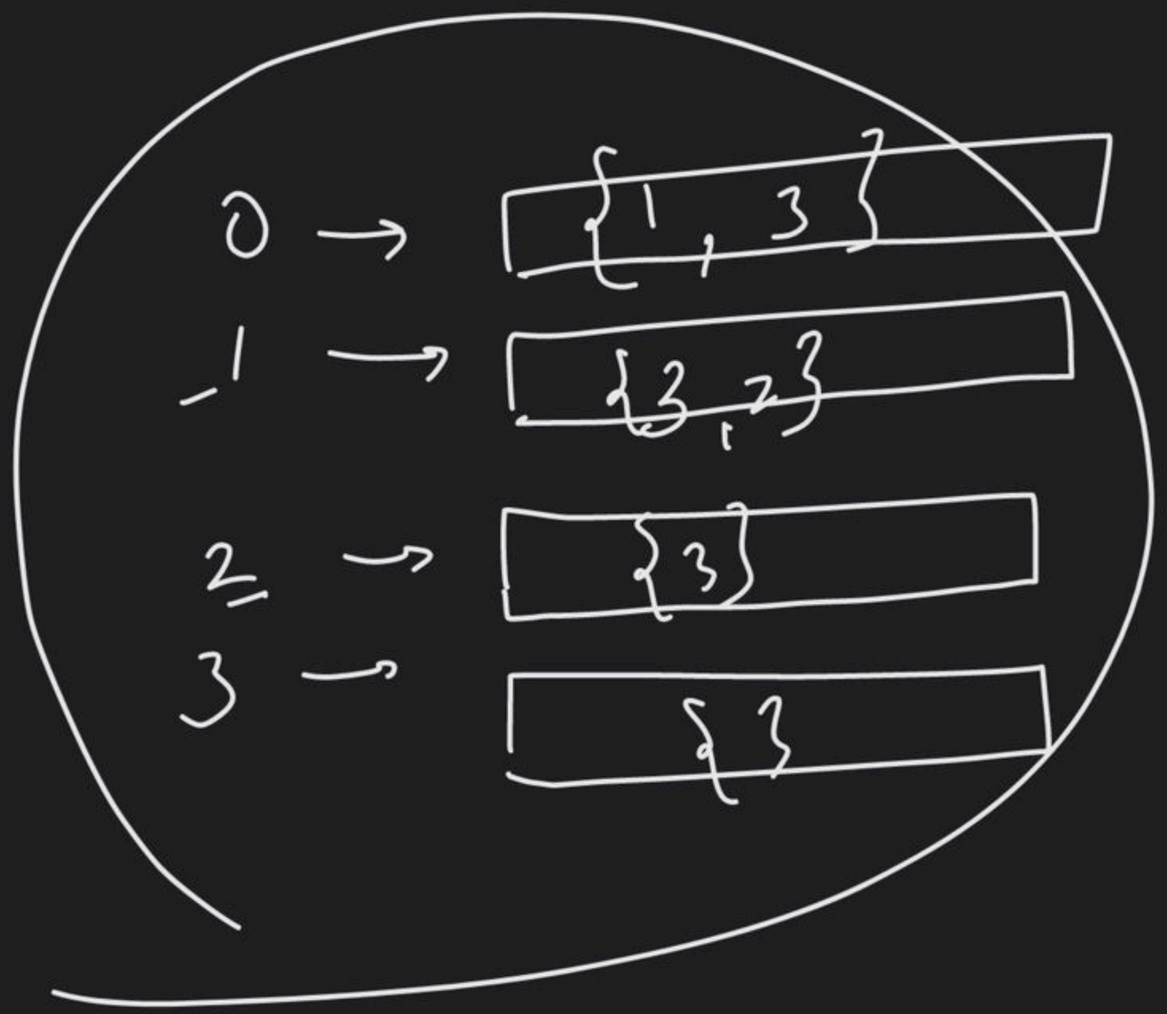


Undirected graph

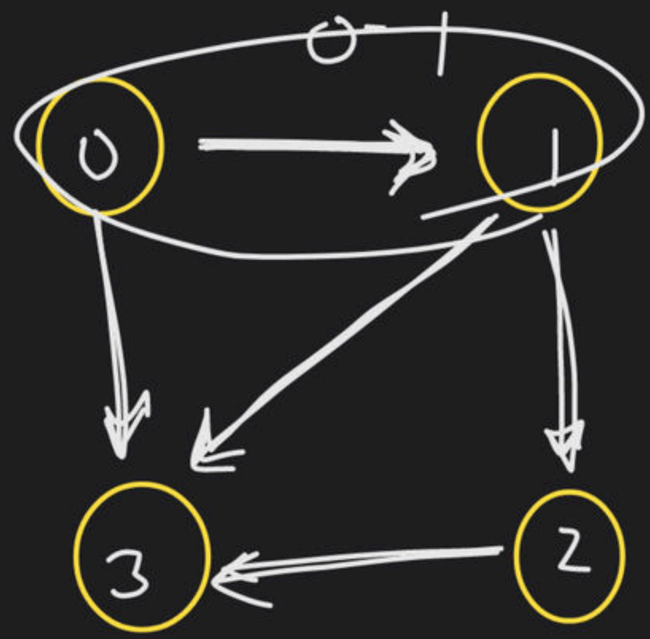

0 1 2
0
1
2
0 1 1
1 0 1
1 1 0
0
1
2
Edge List
0 - 1
1 - 0
2 - 0
0 - 2


i/p →
Edge list
0 - 1
1 - 0
2 - 0
0 - 2
1 - 2
2 - 1
u = 0
v = 1
adj[u][v] = 1

Adjacency List :->



n = 4

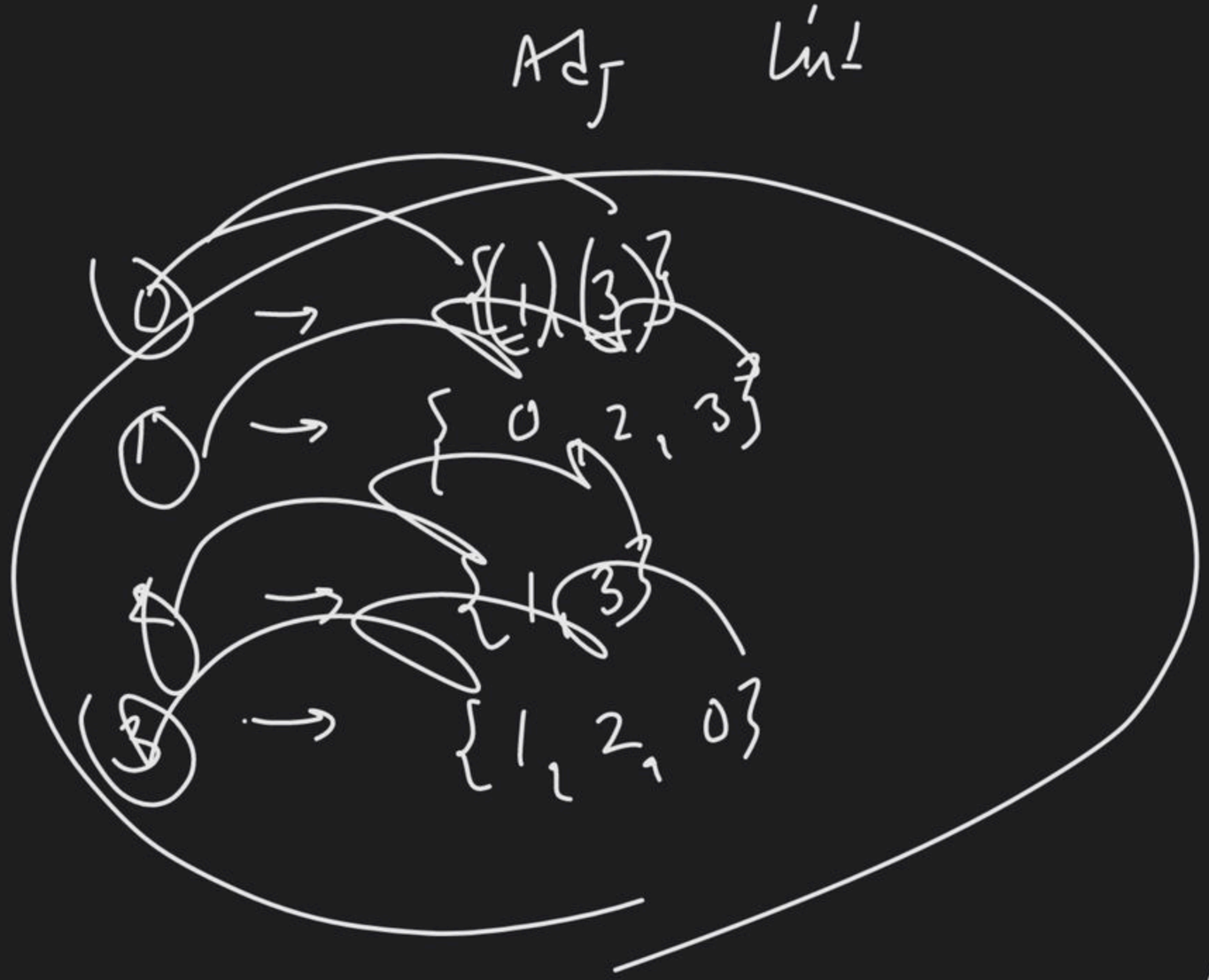


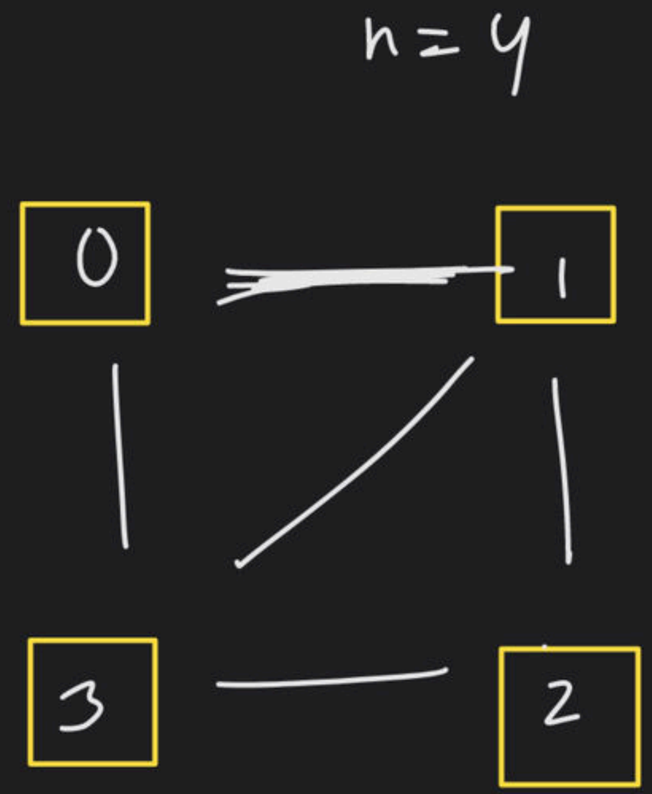
List of Edges :-

[0 → 1]
[1 → 3]
[1 → 2]
[2 → 3]
[0 → 3]

0 — 1

1 — 2





Edges:-

0 → 1	1 → 3	0 → 3
1 → 0	3 → 1	3 → 0
1 → 2	2 → 3	
2 → 1	3 → 2	

Adj List

node dist

0 → {(1, 3) (3, 8)}

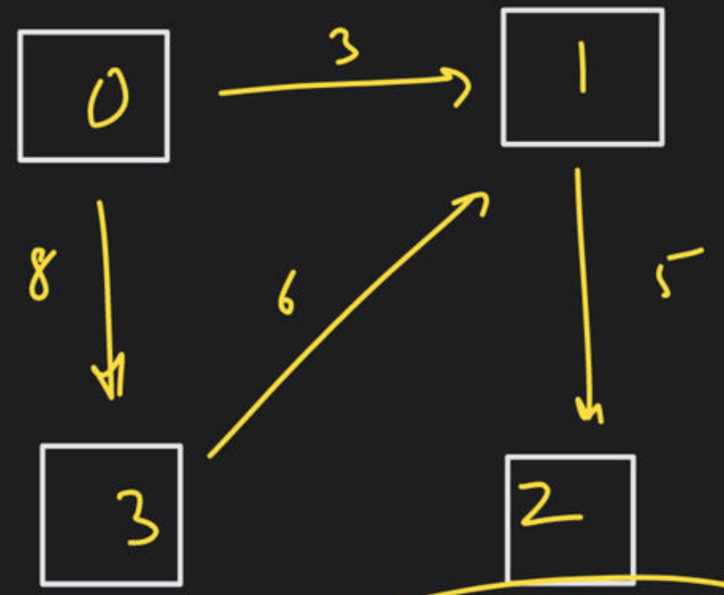
1 → {(2, 5)}

2 → {}

3 → {(1, 6)}

0, ∞ ← distance

k = 1



edge

0 —3→ 1

3 —6→ 1

1 —5→ 2

0 —8→ 3

Adj list

0 → 1, 2

1 → 0, 2

2 → 0, 1

0 → 1

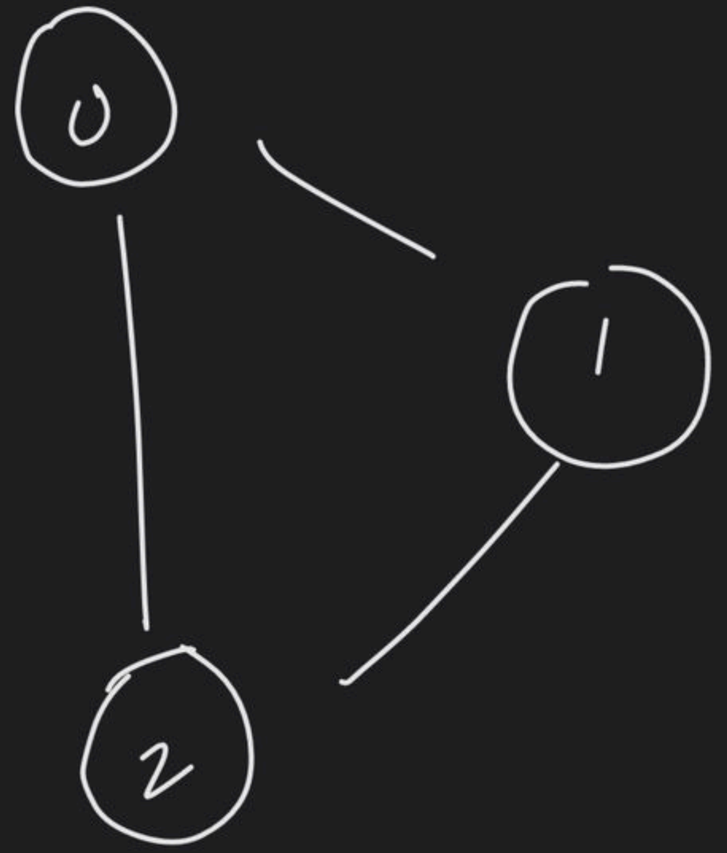
1 → 0

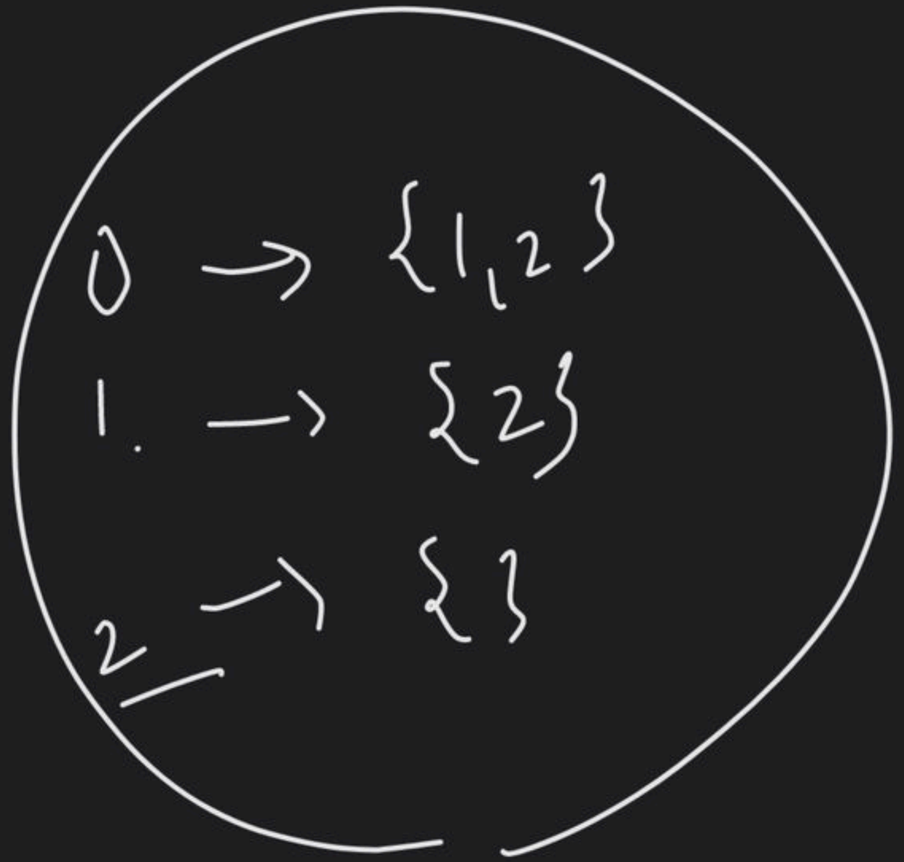
1 → 2

2 → 1

0 → 2

2 → 0



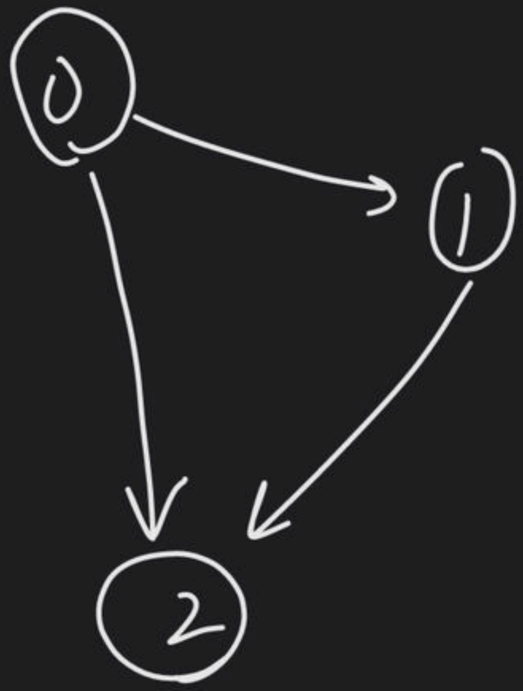


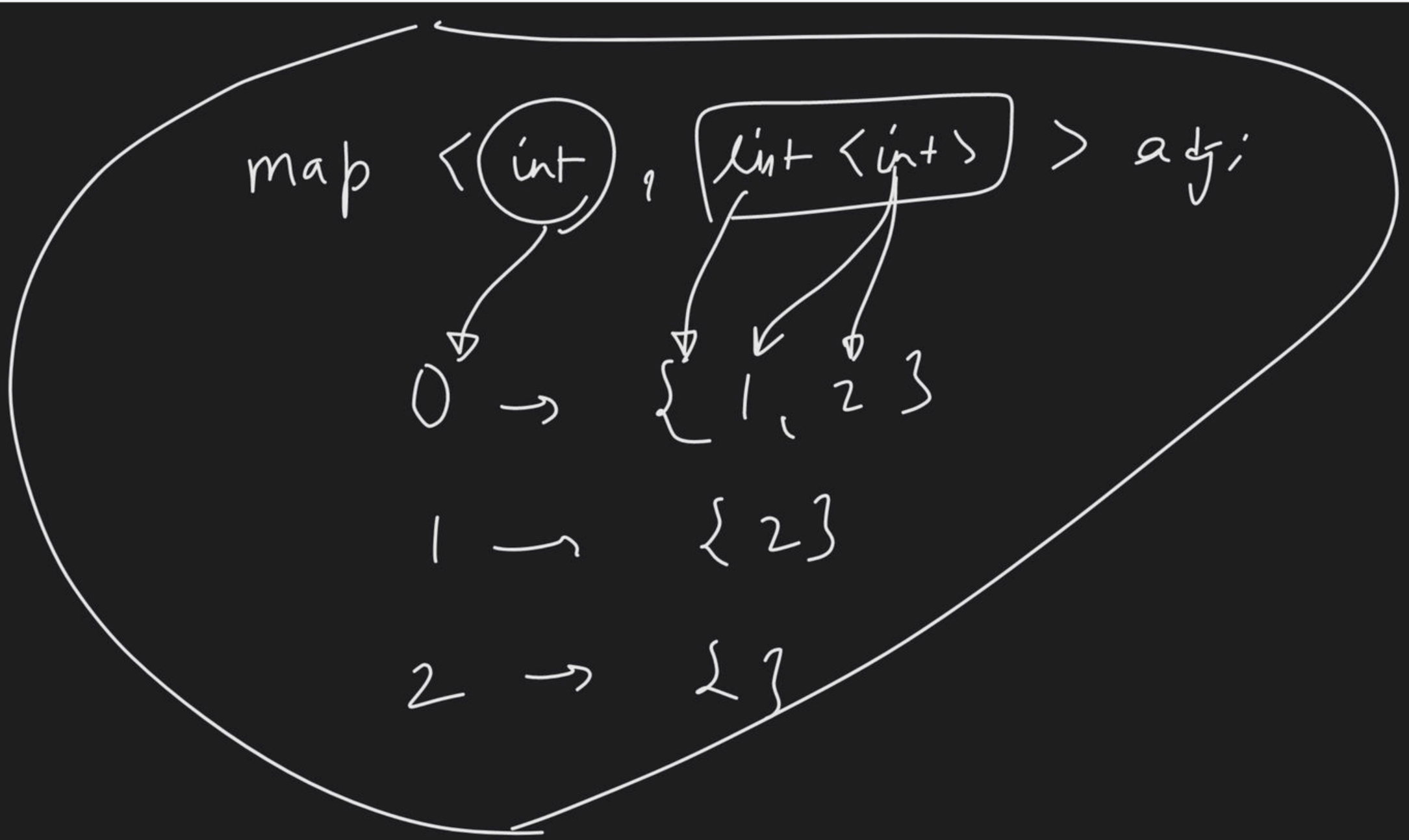
Edges

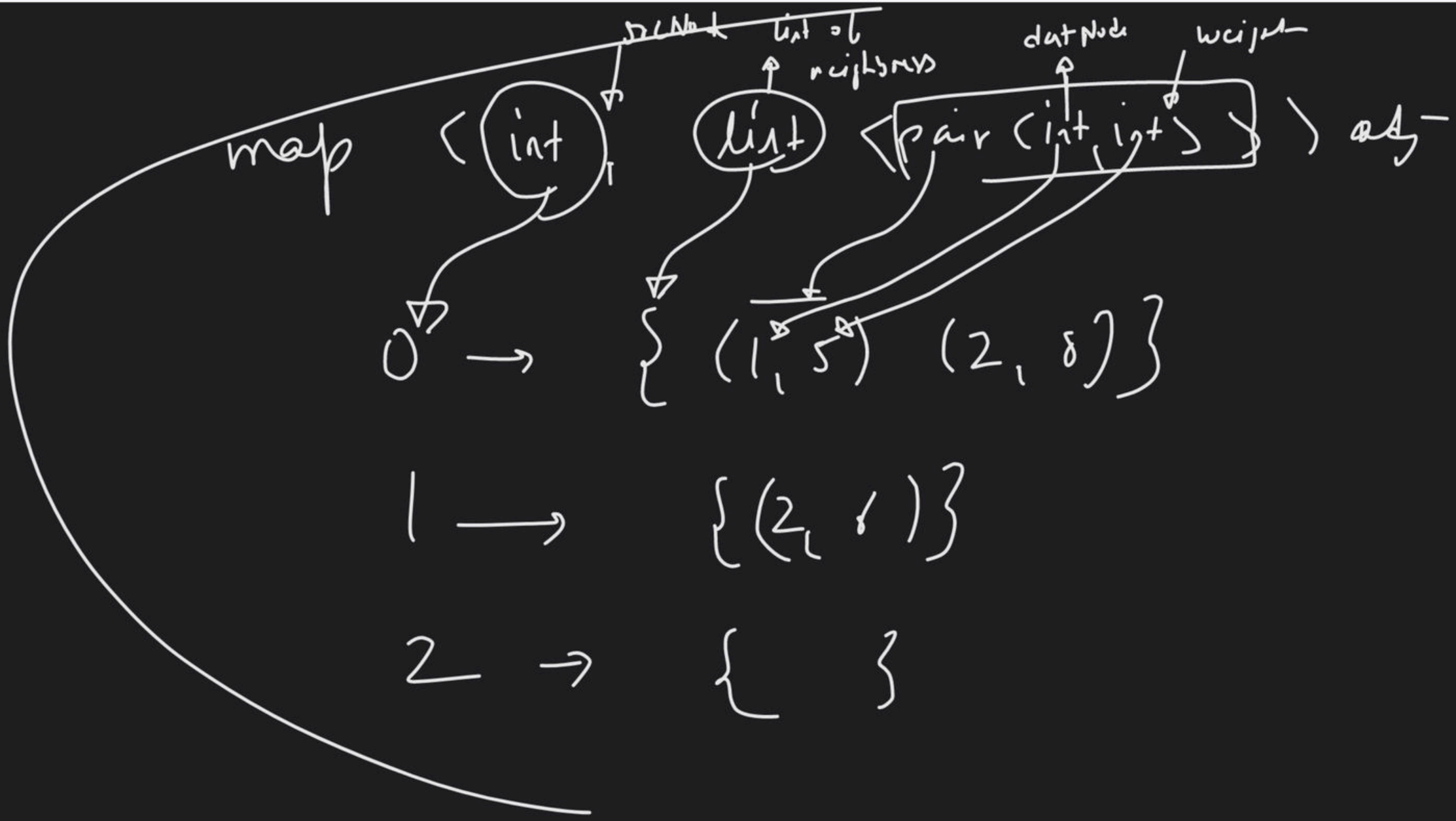
0 → 1

1 → 2

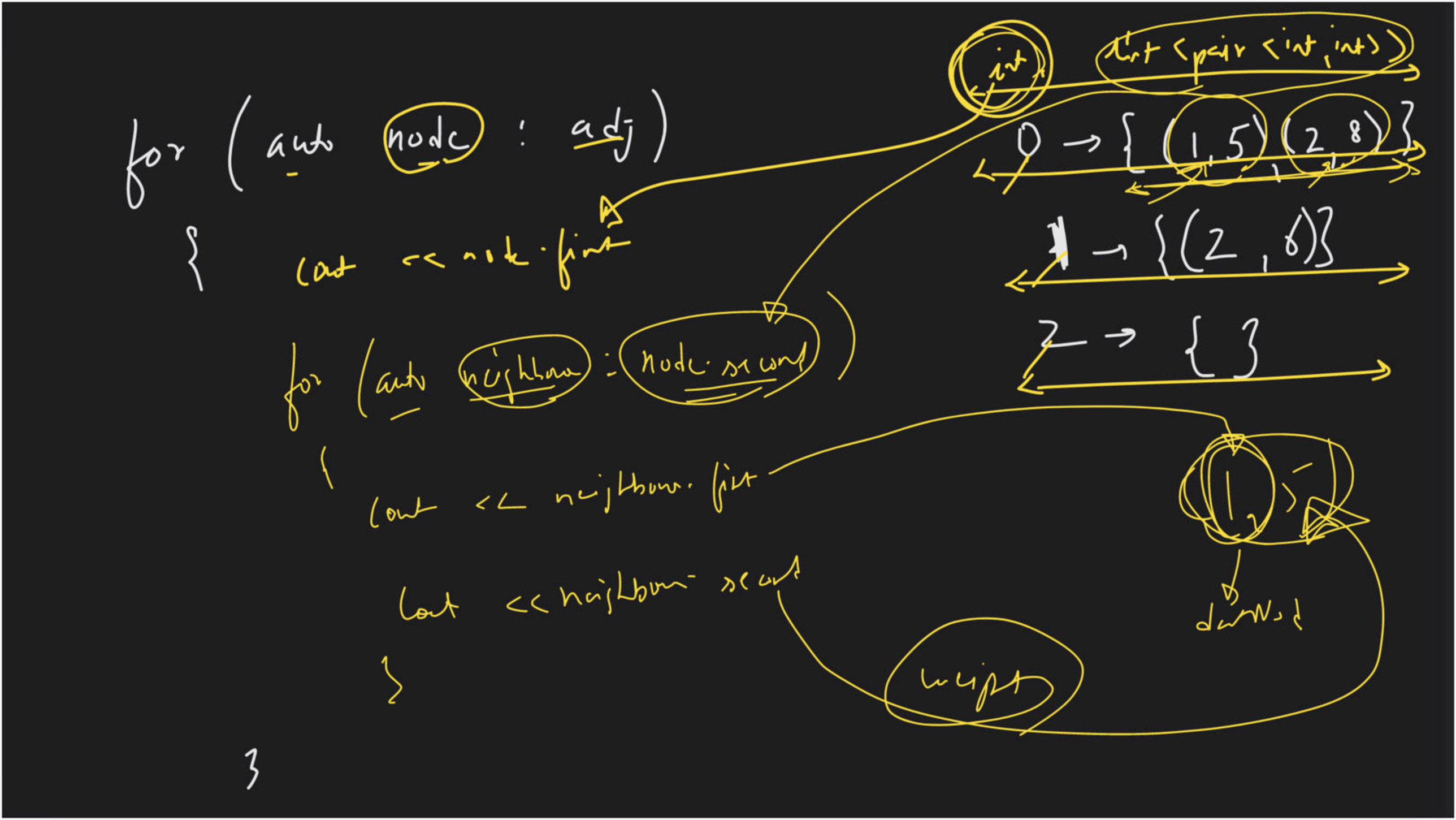
0 → 2

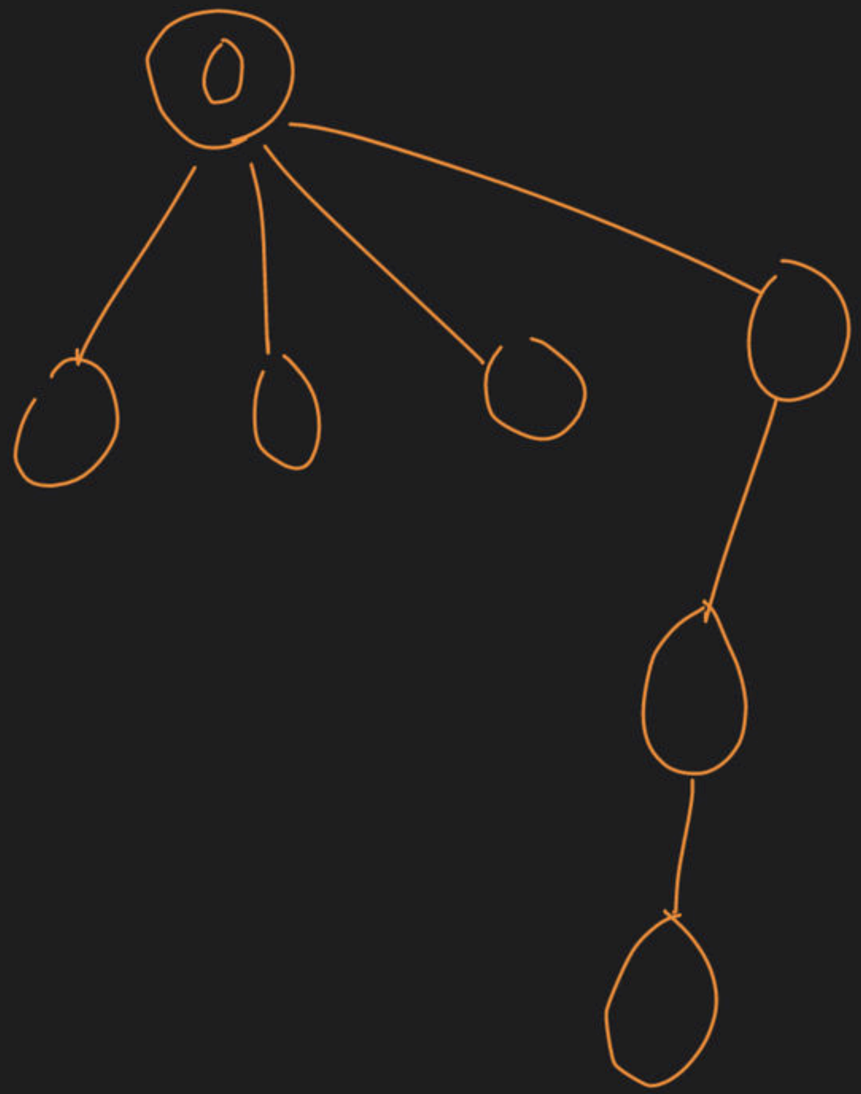


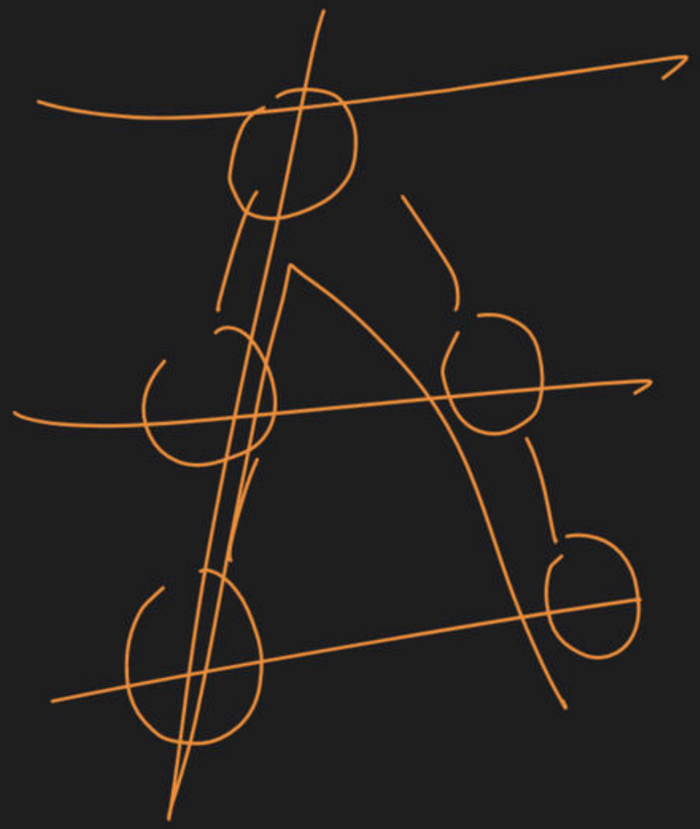

map < int , list < int > > adj;
0 → { 1, 2 }
1 → { 2 }
2 → { }

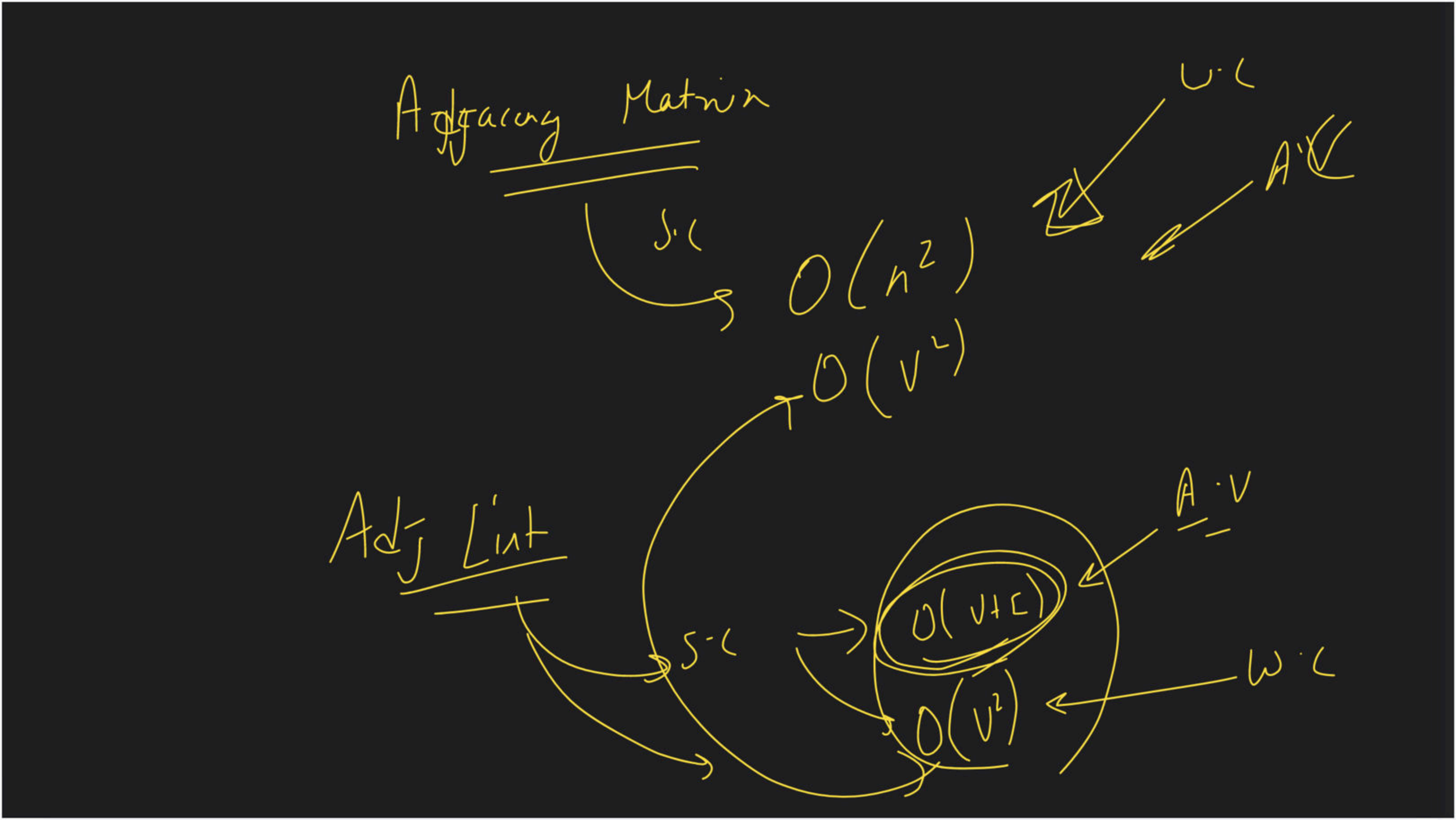

map
int
list
pair (int, int)
adj
0 → { (1, 5) (2, 8) }
1 → { (2, 1) }
2 → { }

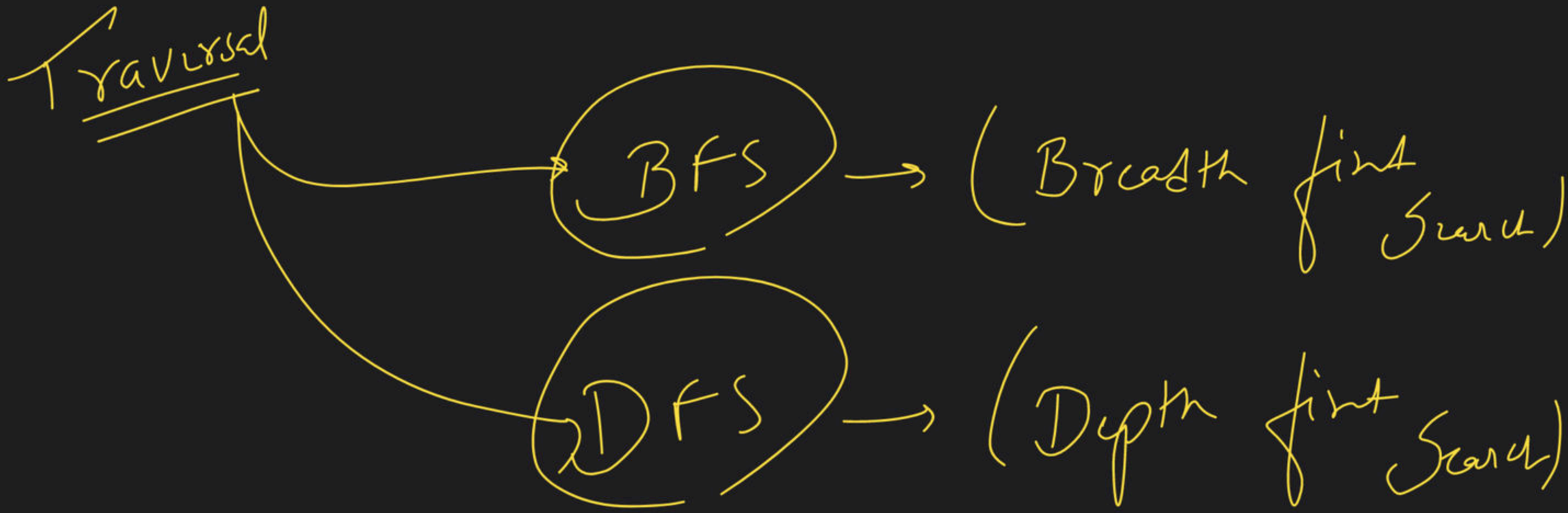
```
for (auto node : adj)
{
    cout << node.first
    for (auto neighbour : node.second)
    {
        cout << neighbour.first
        cout << neighbour.second
    }
}
```

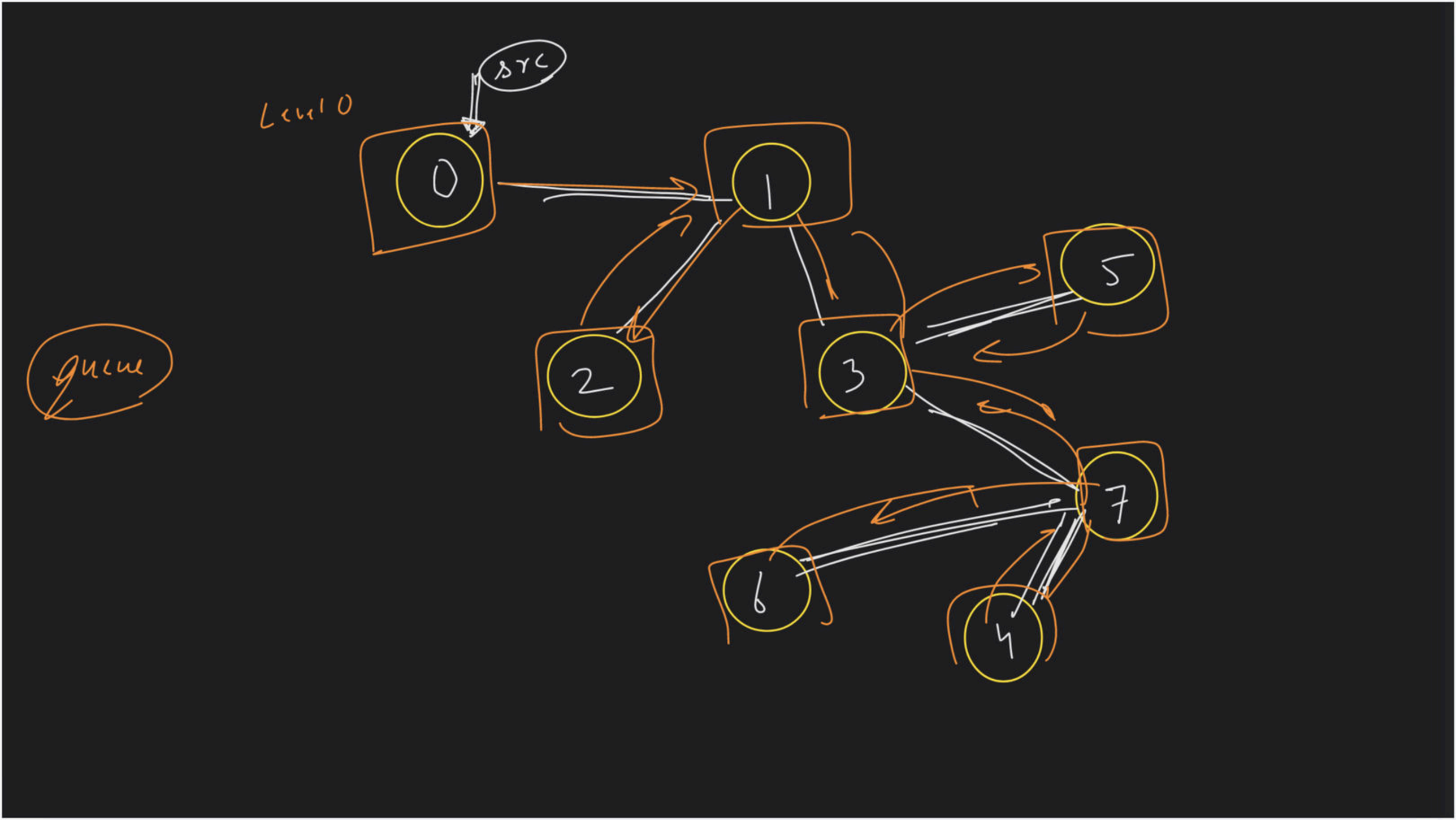







Adjacency Matrix
S.C
O(n²)
O(V²)
U.C
A.C
Adj List
S.C
O(V+E)
O(V²)
A.V
W.C


Traversal
BFS
(Breadth first Search)
DFS
(Depth first Search)


src
Level 0
0
1
2
3
5
7
6
4
queue

Queue 0 1 2 3 5 7 4 6

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

src = 0
Q.push(0)
Vis[0] = True
} -> initial steps

while (!Q.empty())
{
I -> fNode
II -> pop
III print fNode
IV -> insert neighbour
}

visited[fNode] = T

Visited

src = 0

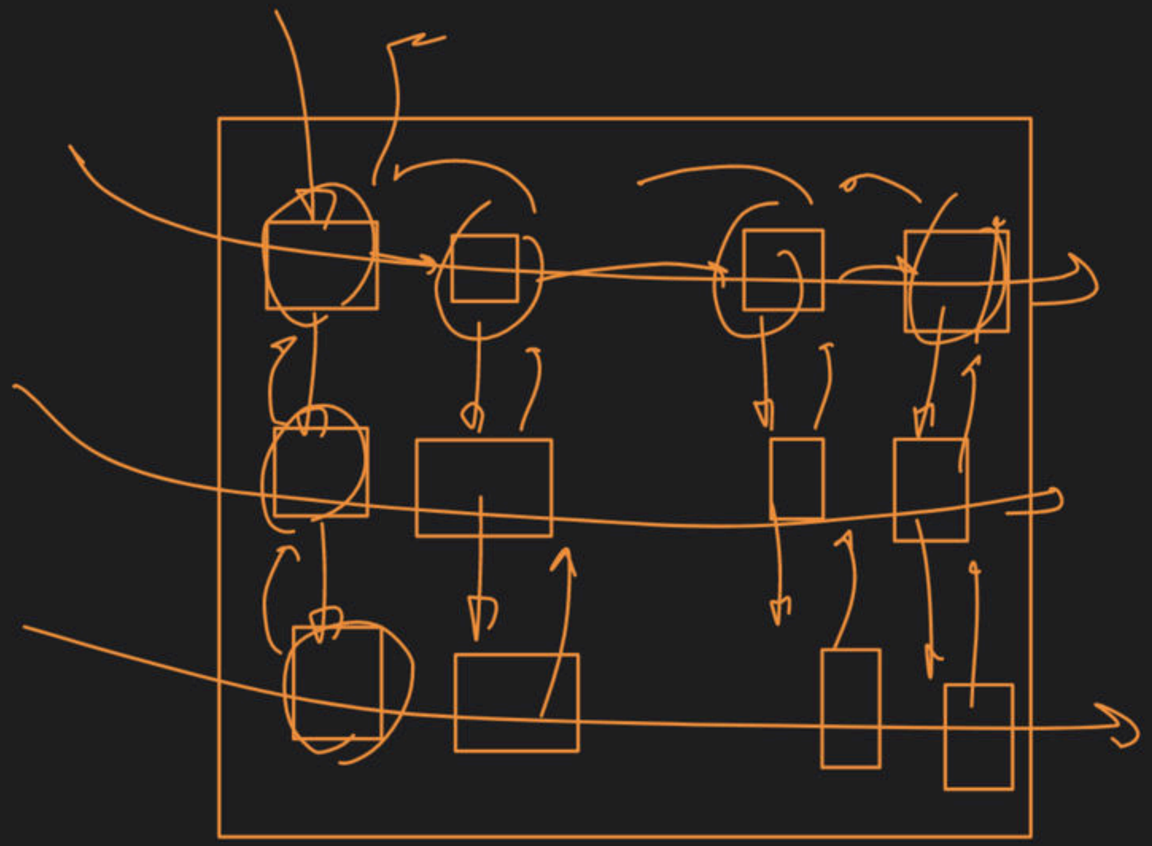
Adj List

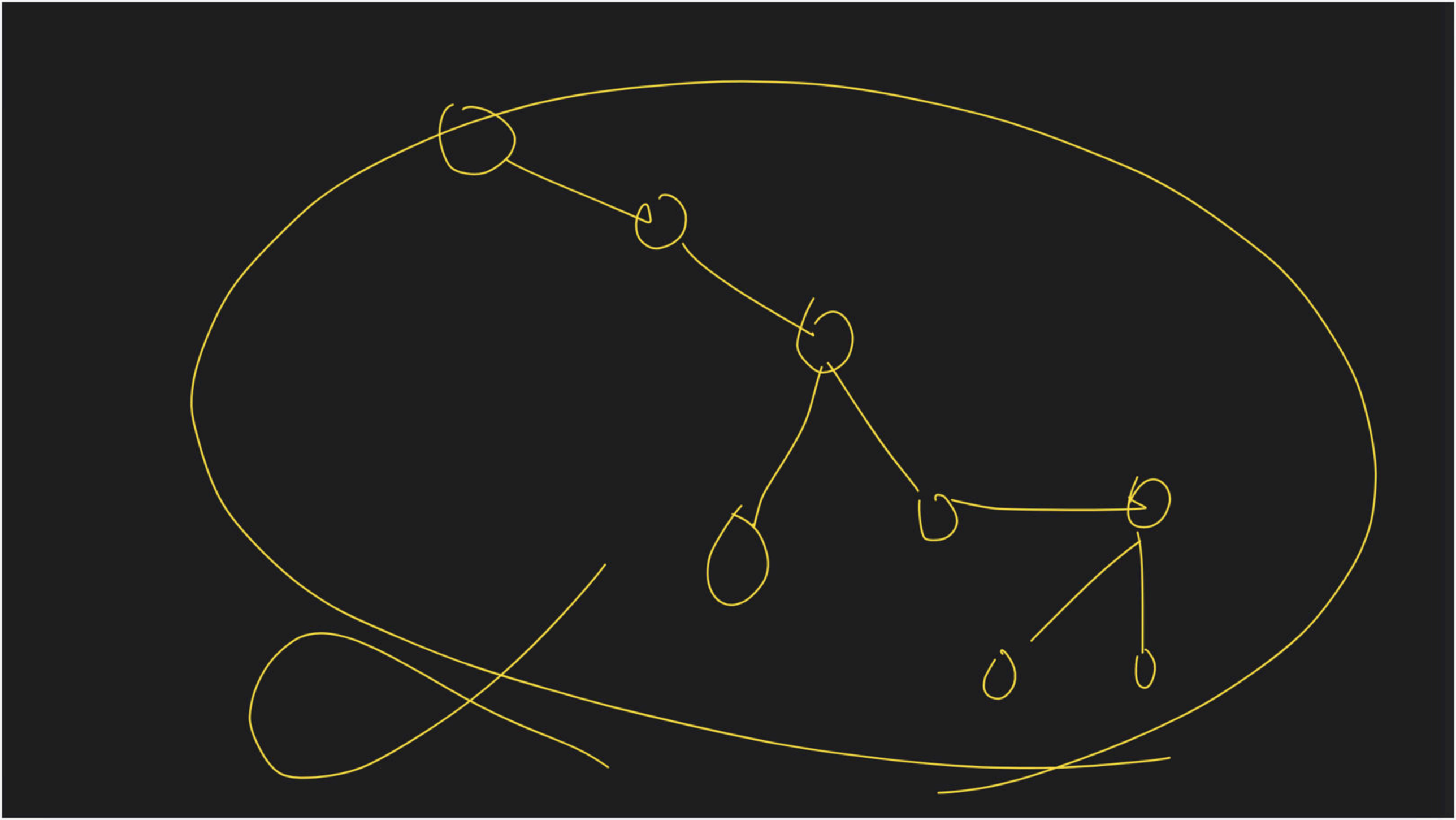
0 -> {1}
1 -> {2, 3, 4, 0}
2 -> {1}
3 -> {1, 5, 7}
4 -> {7}
5 -> {3}
6 -> {7}
7 -> {3, 4, 6}

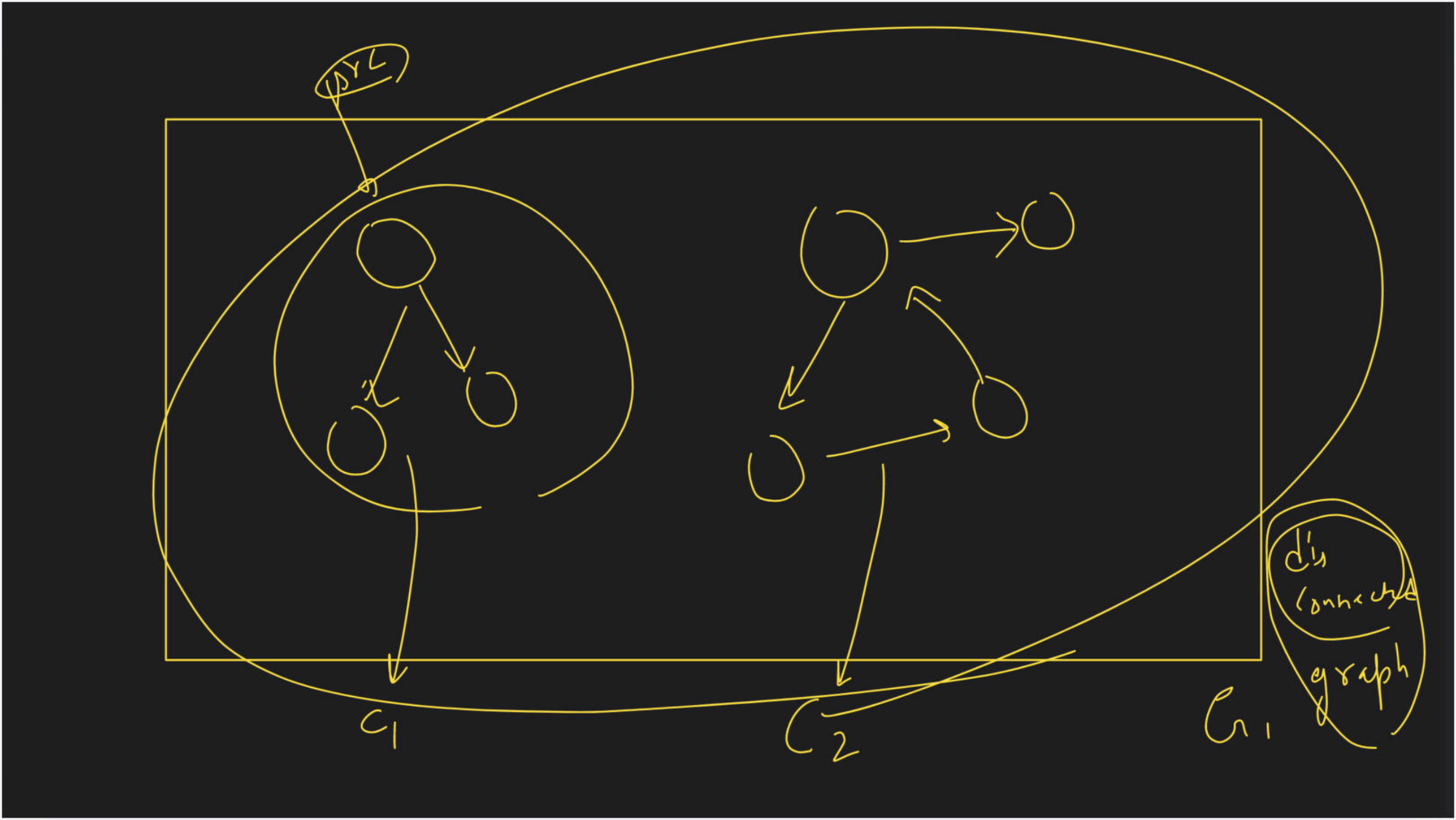
0 -> F T		3 -> F T	
1 -> F T		4 -> F T	
2 -> F T		5 -> F T	
		6 -> F T	
		7 -> F T	

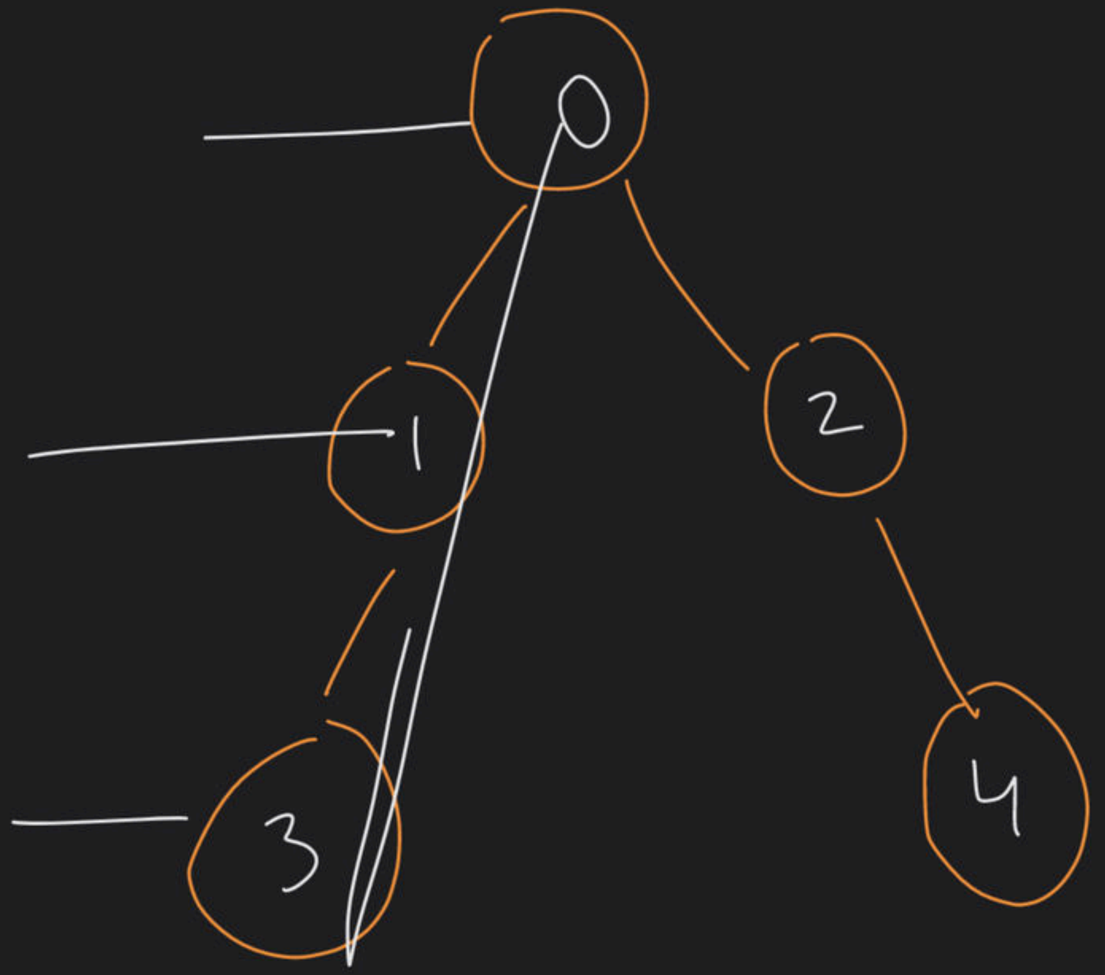
Edge List

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






DYL
C_1
C_2
C_n
disconnected
graph

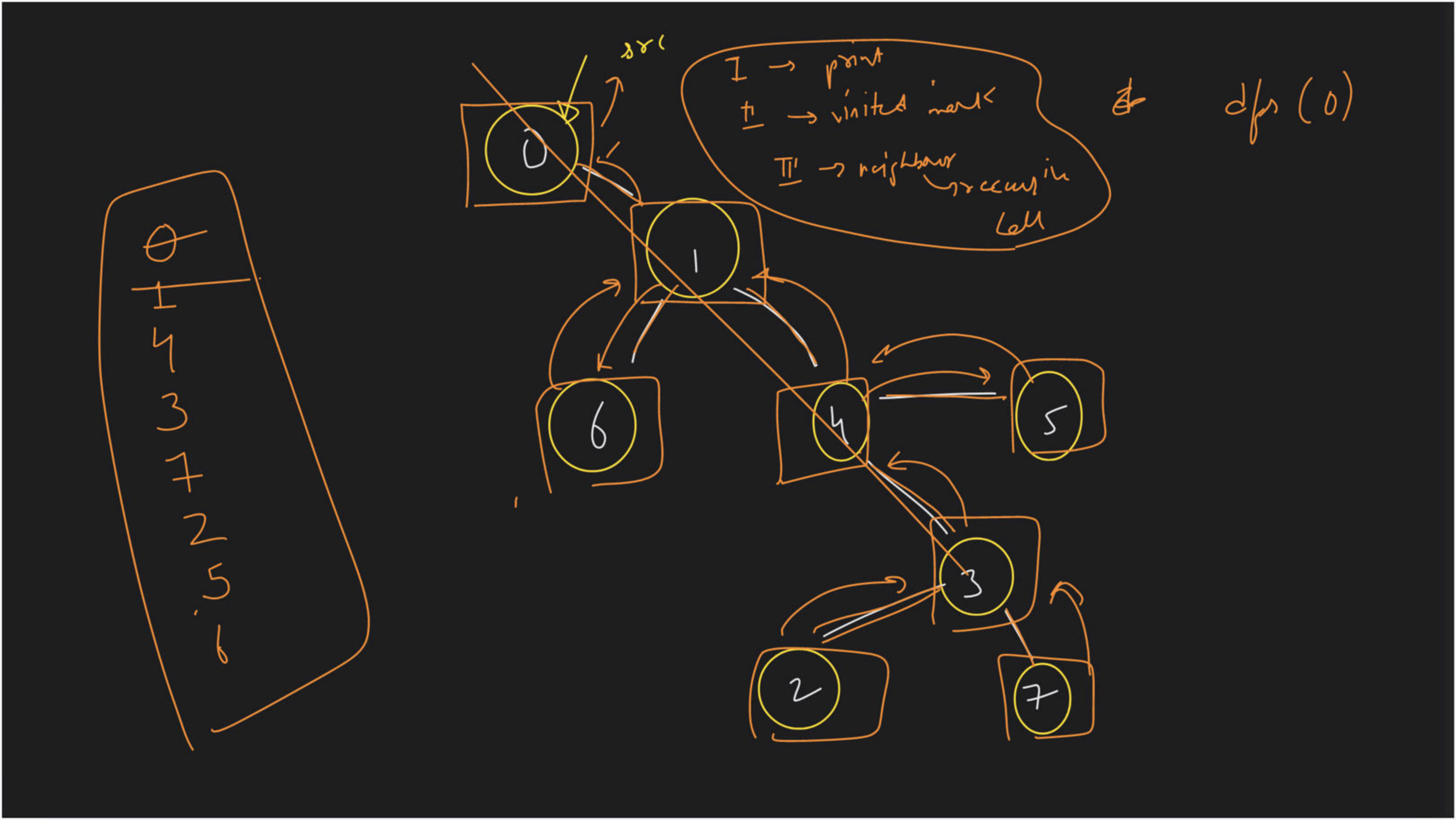


BFS 0 1 2 3 4

DFS

0 1 3 2 4

0 2 4 1 3


I → print
II → visited mark
III → neighbour processing
dfs (0)
0
1
3
7
2
5
6
0
1
2
3
4
5
6
7