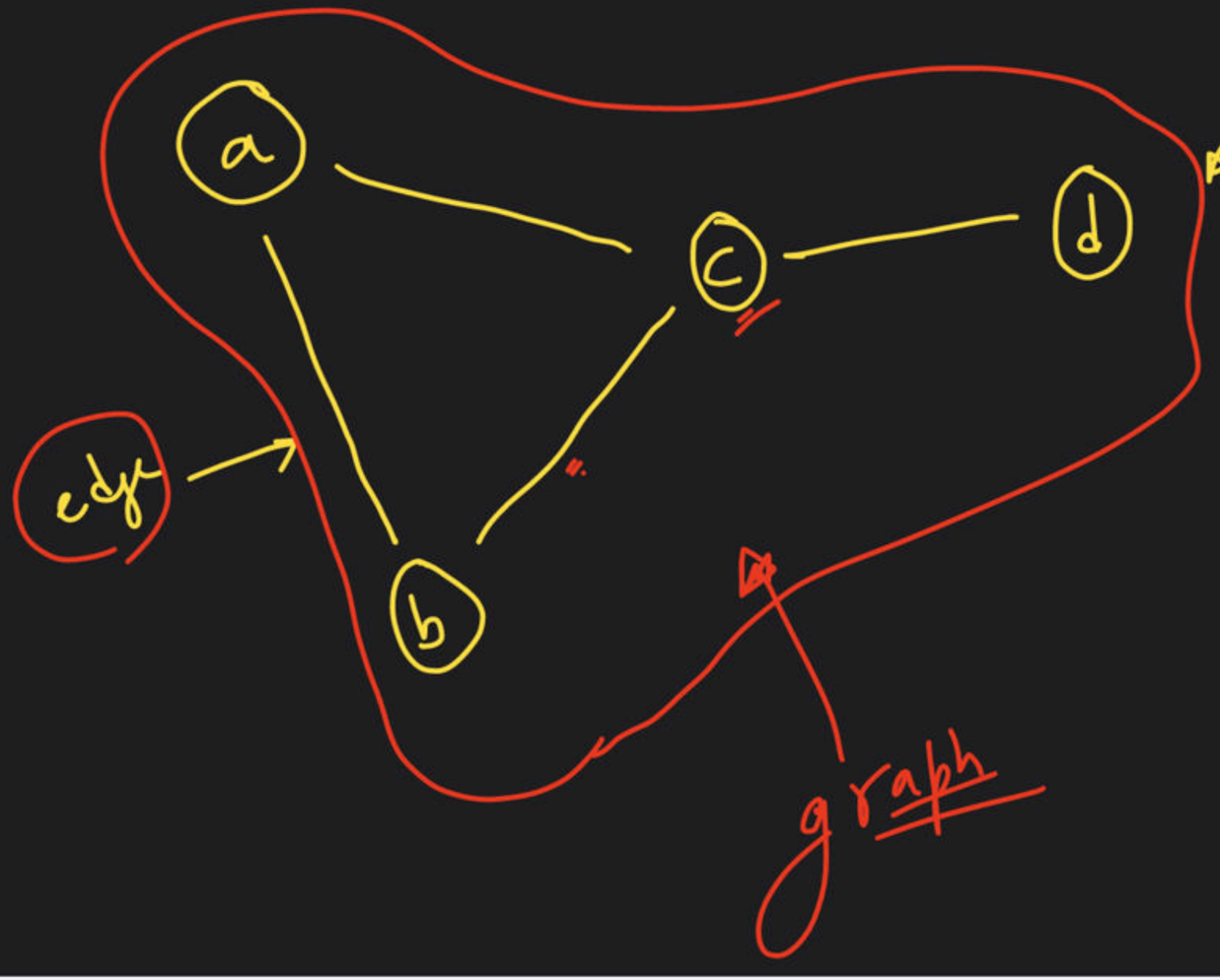
Graphs Class - 1

Special class

Love Babbar • Jan 24, 2024

→ Graph → D.S → Edges
→ Node



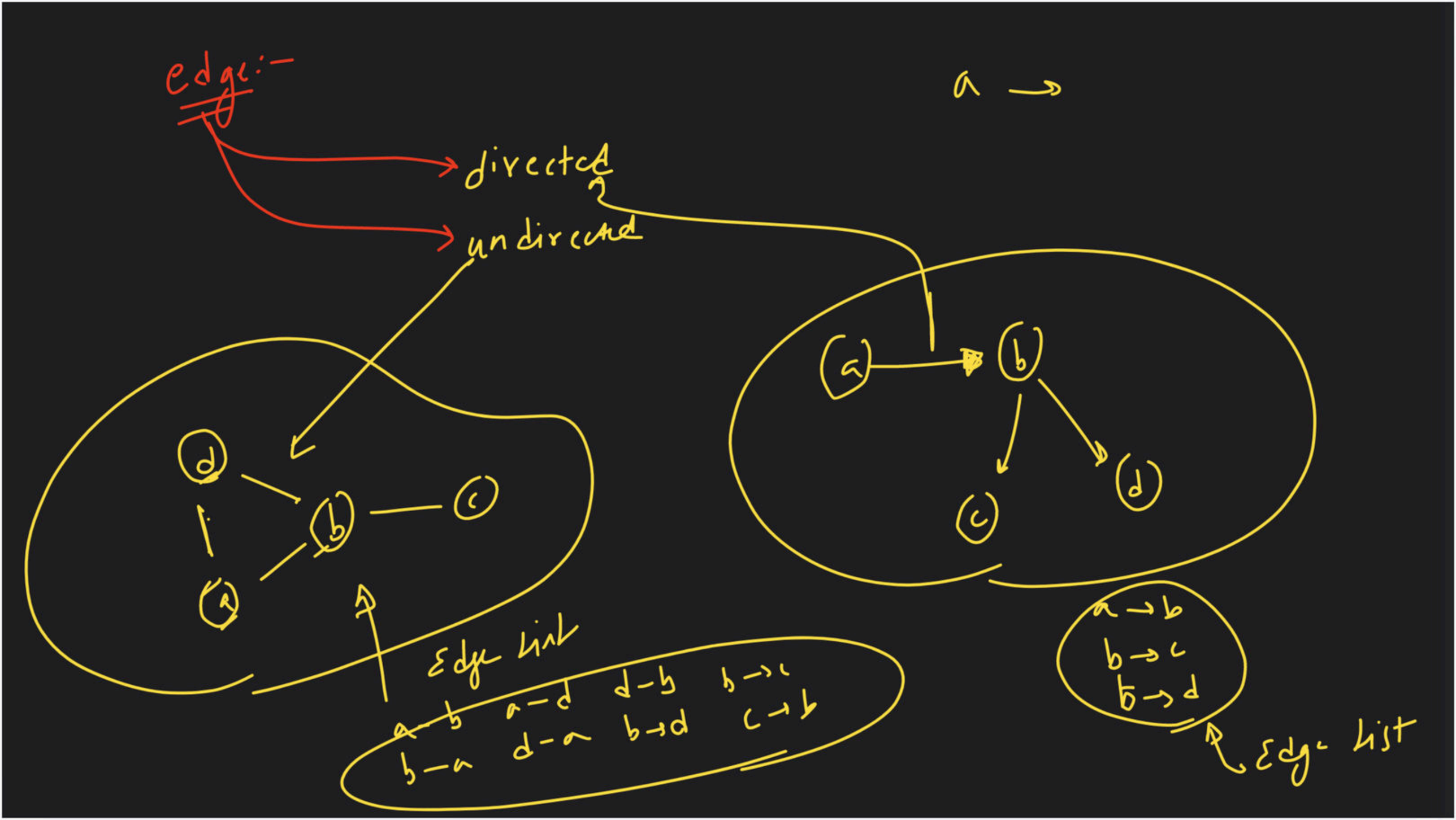
int
char

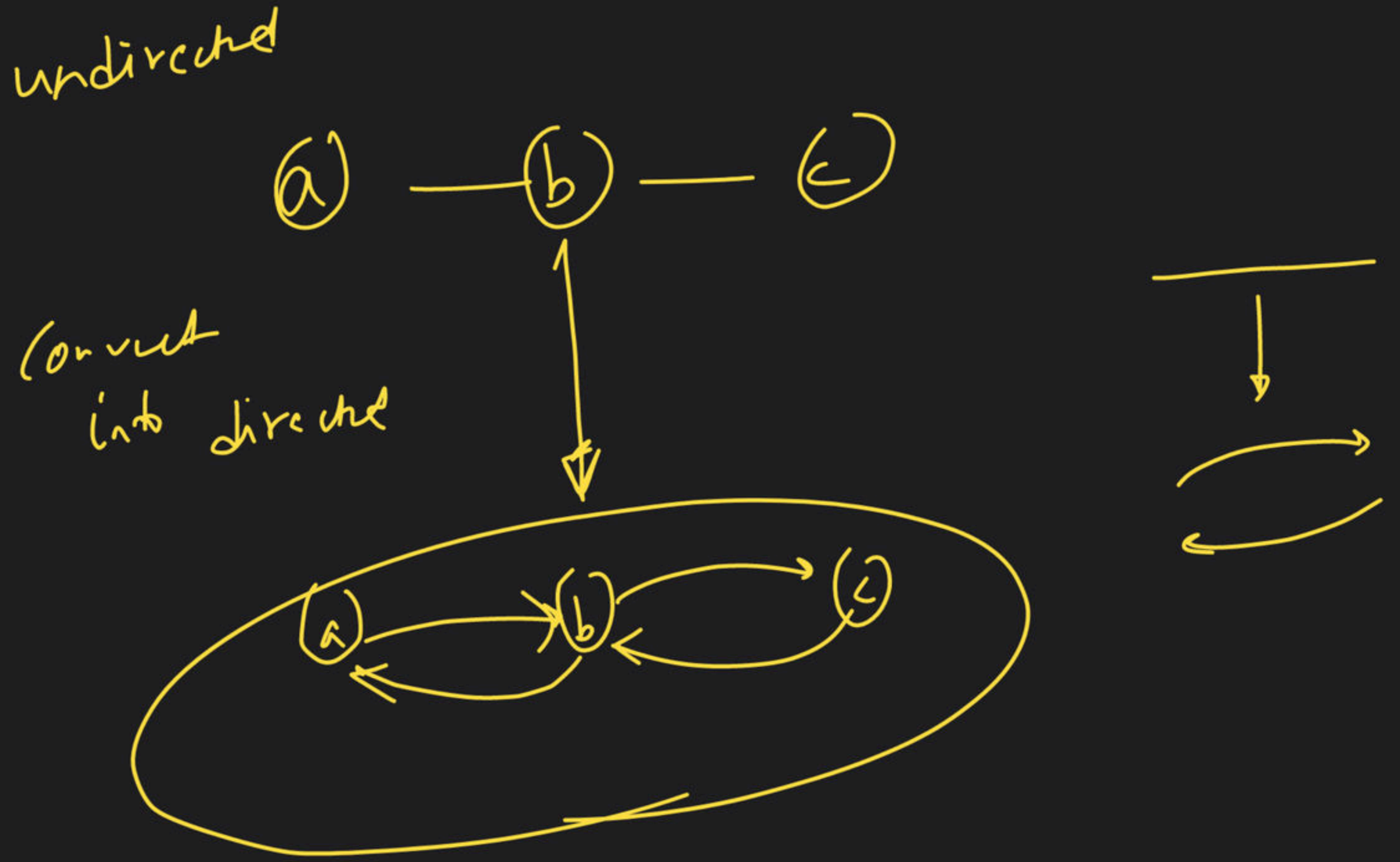
node

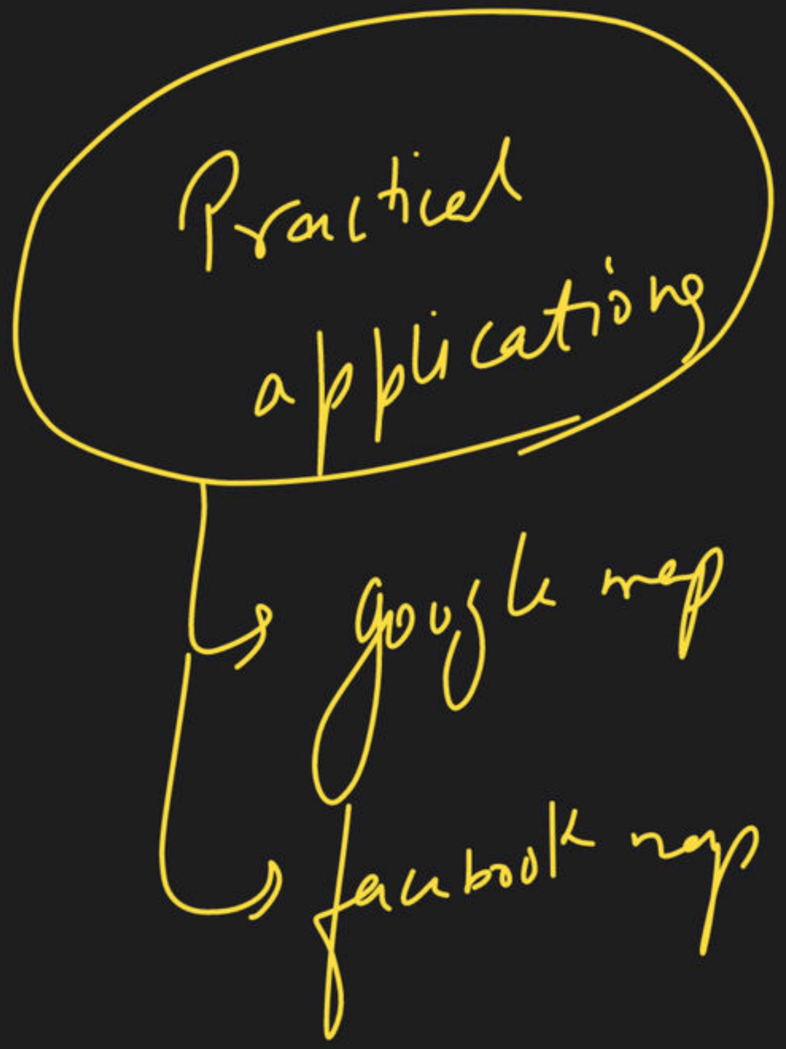
d

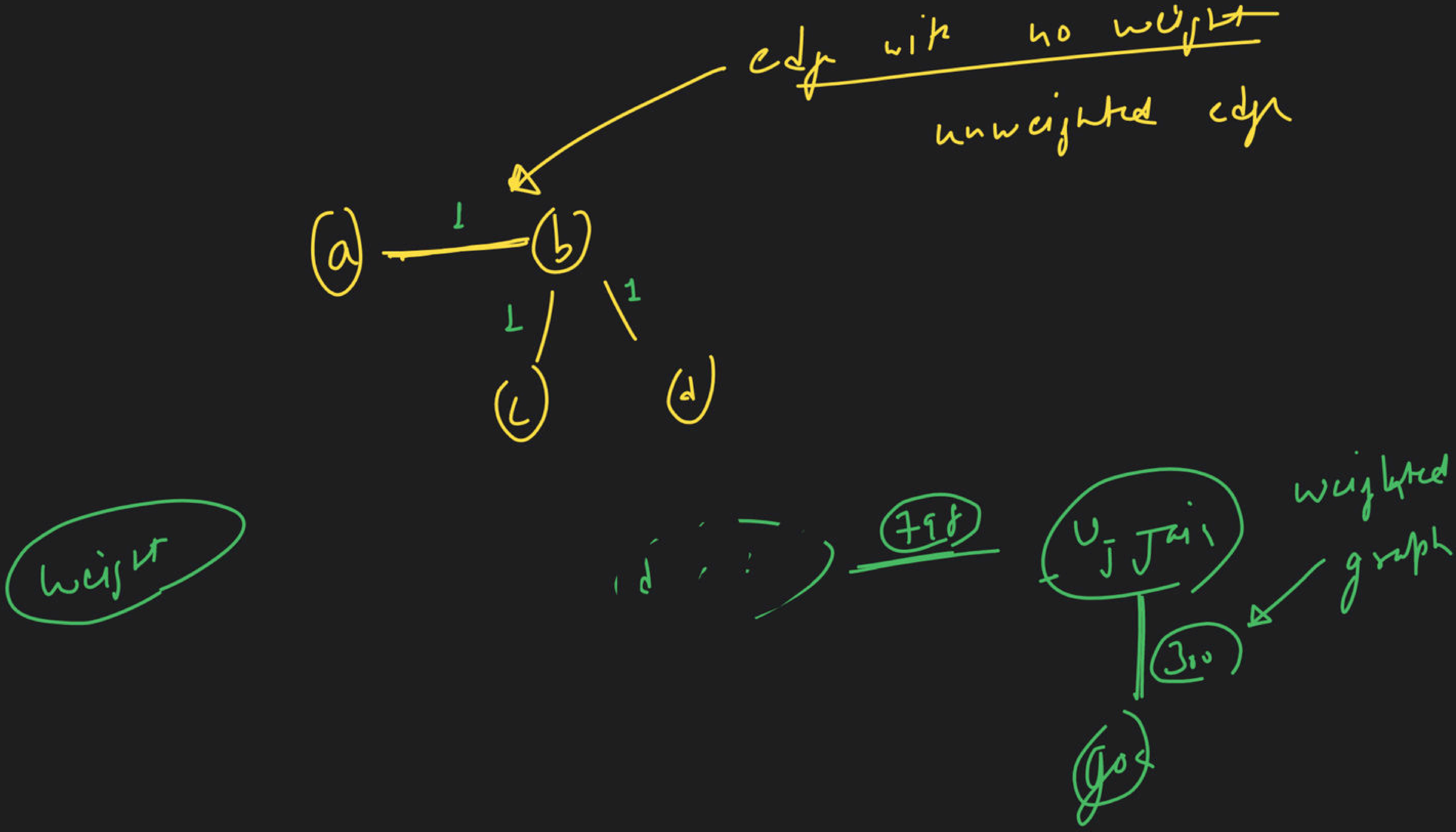
int
char
str

class

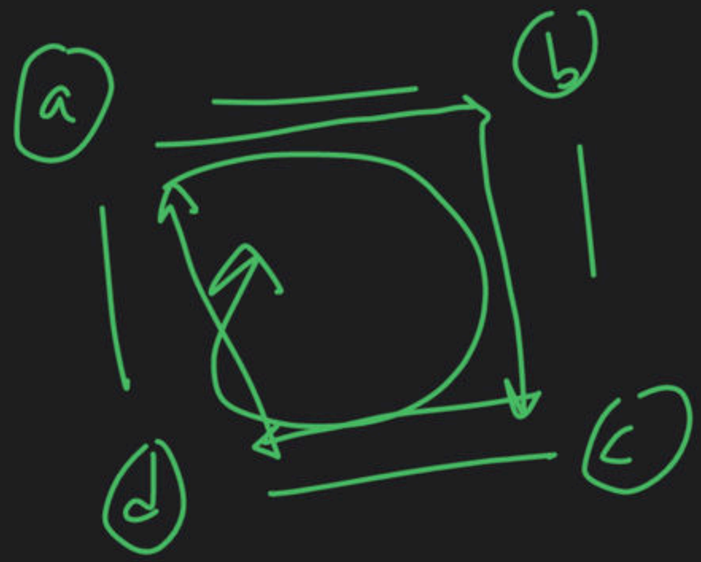

edge:-
directed
undirected
a →
a
b
c
d
a
b
c
d
Edge list
a-b a-d d-b b→c
b-a d-a b→d c→b
a→b
b→c
b→d
Edge list

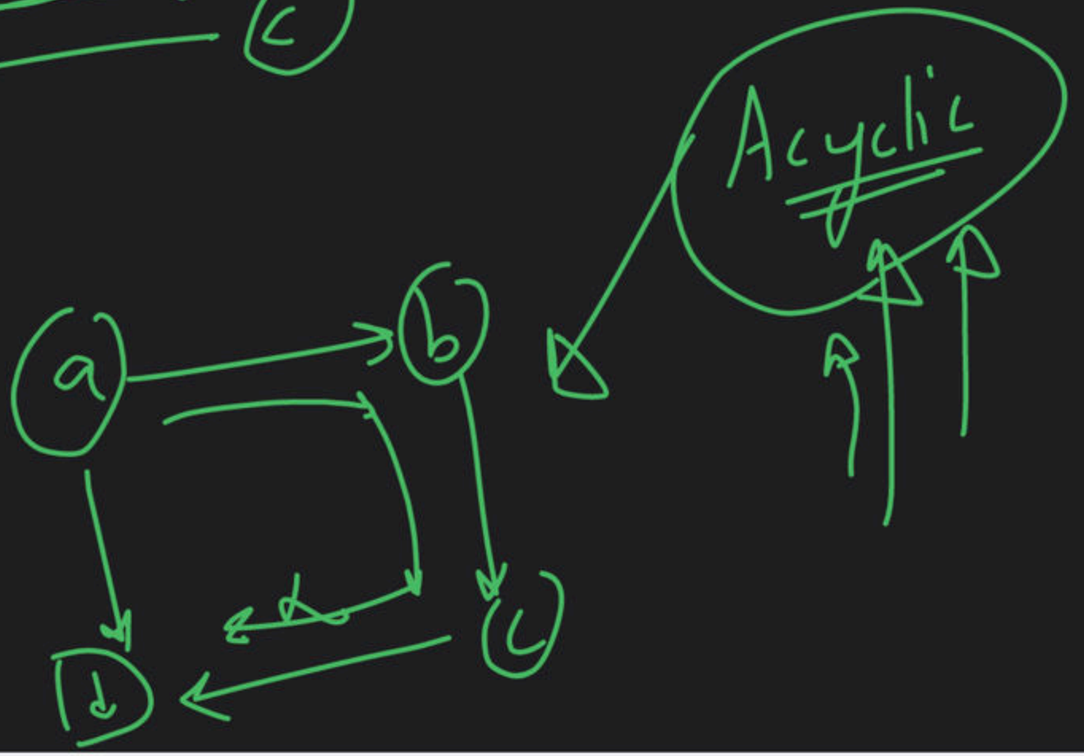

undirected
a
b
c
Convert
into directed
a
b
c


Practical
applications
google map
facebook map


edge with no weight
unweighted edge
a
b
c
d
1
1
1
weight
798
300
weighted graph

Cyclic graph





degree

degree(a) = 3

degree(b) = 2

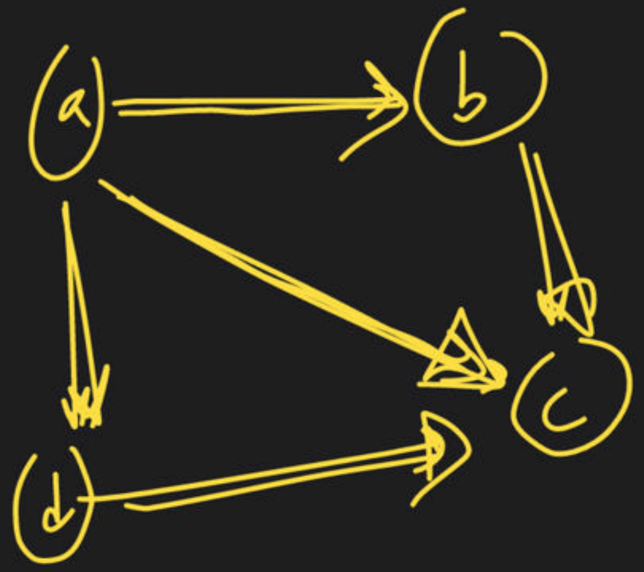
degree(d) = 2

degree(e) = 3

Directed graph

Indegree

Outdegree



Indegree(a) = 0

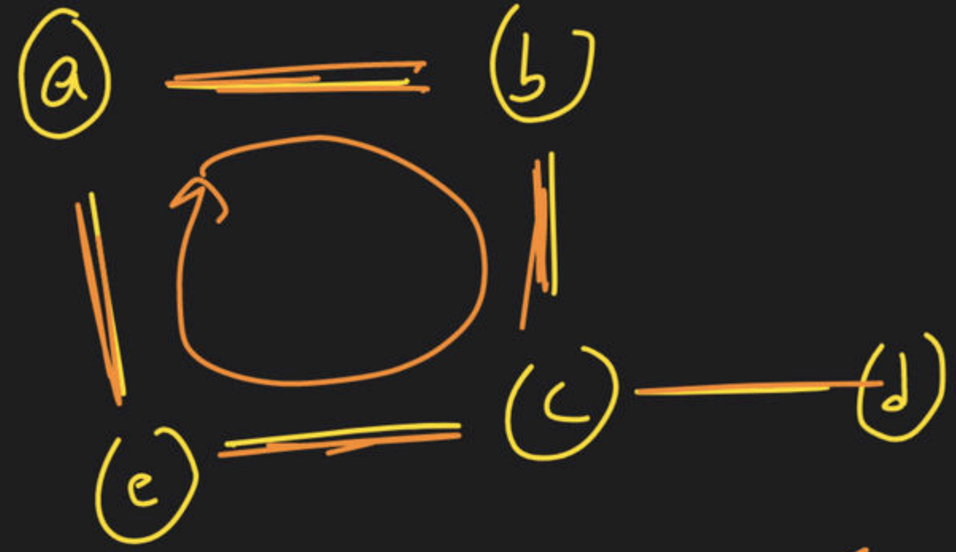
Outdegree(c) = 3

Indegree(c) = 2

Outdegree(d) = 0

Path :-

a - b - c - c - c



→ path

a - b - c - d

a - b - c - e - a

graph

→ edge

→ node

→ undirected

→ directed

→ weighted

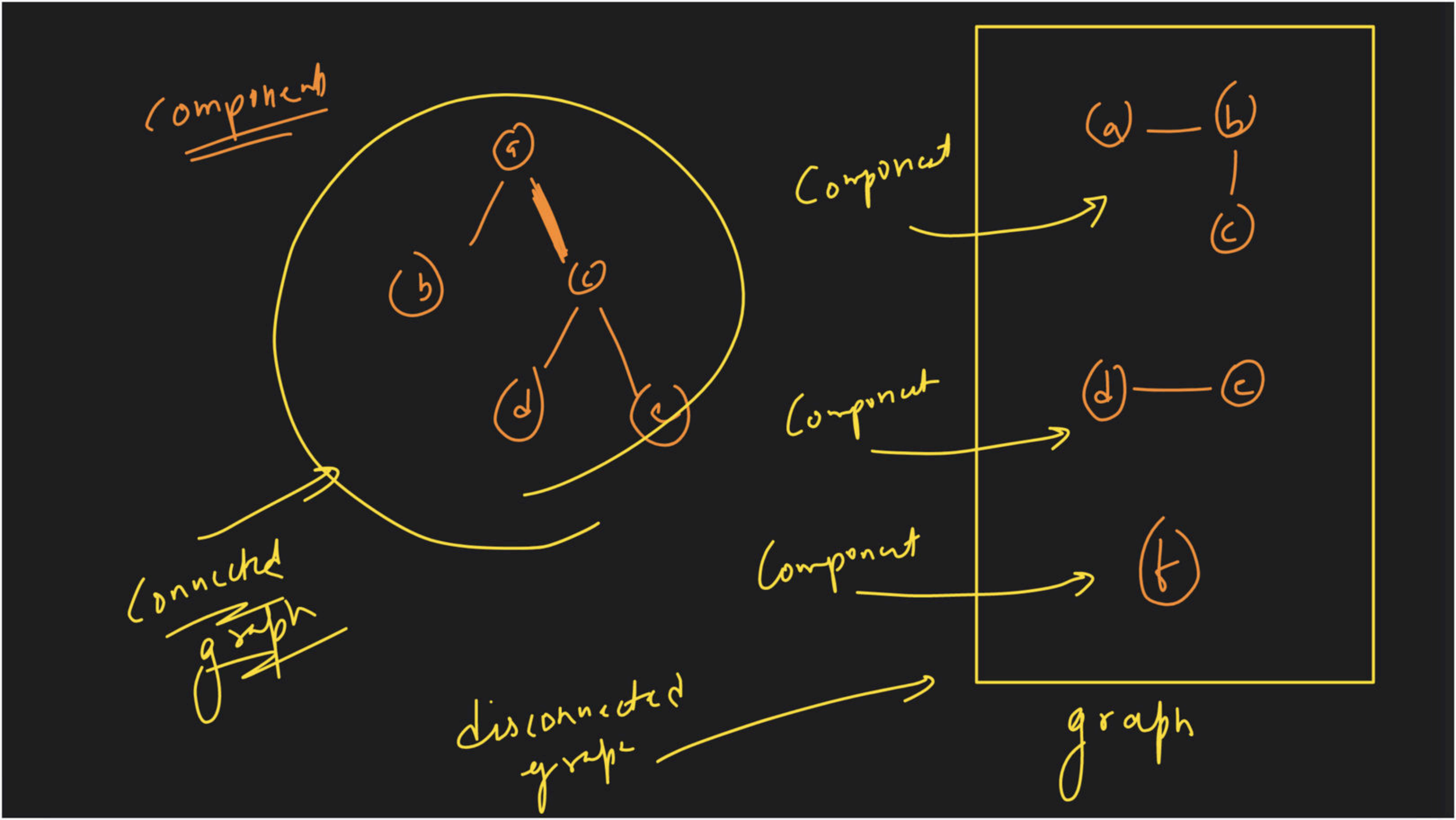
→ unweighted

→ degree

→ indegree

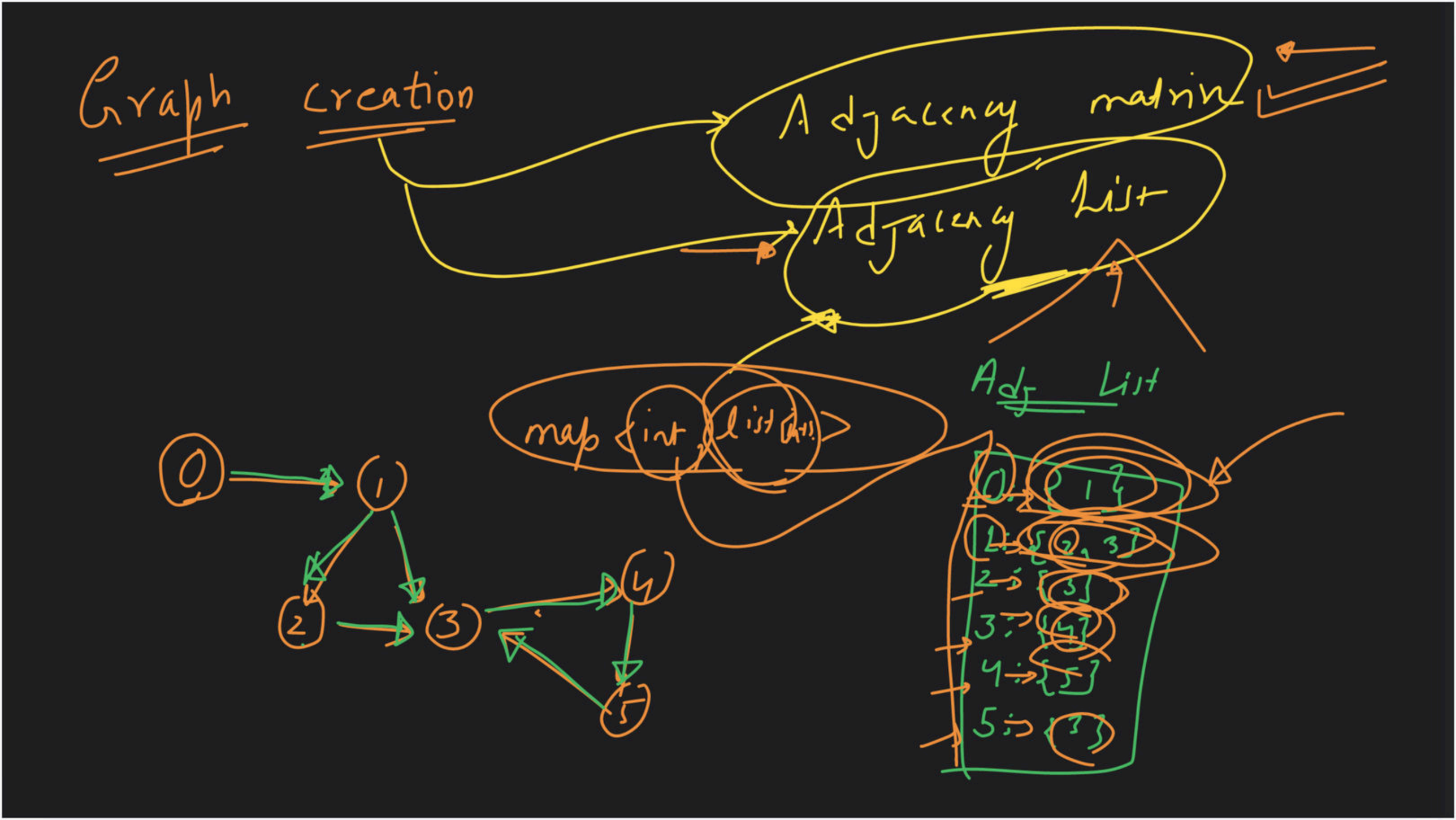
→ outdegree

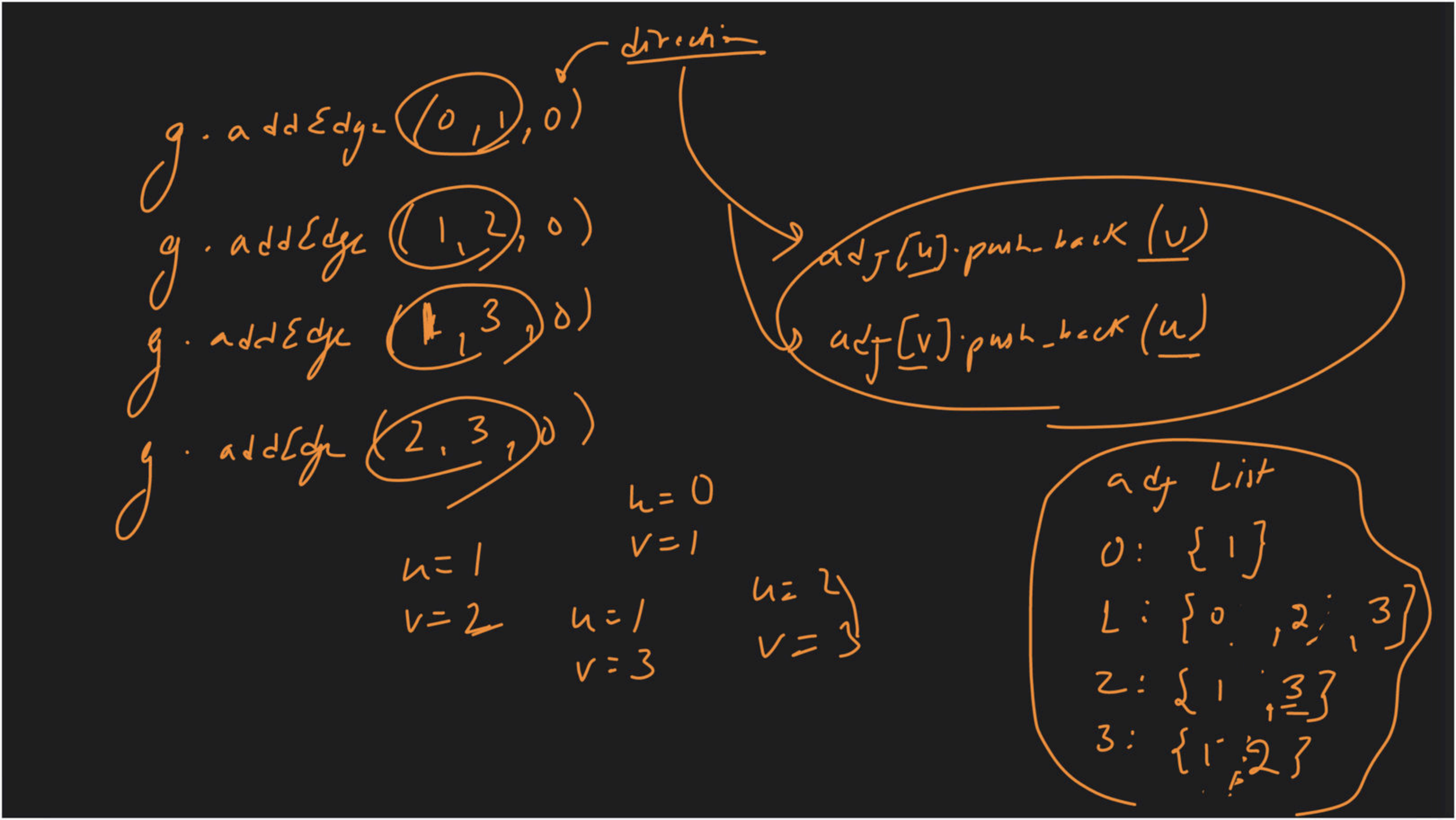
→ cyclic / Acyclic

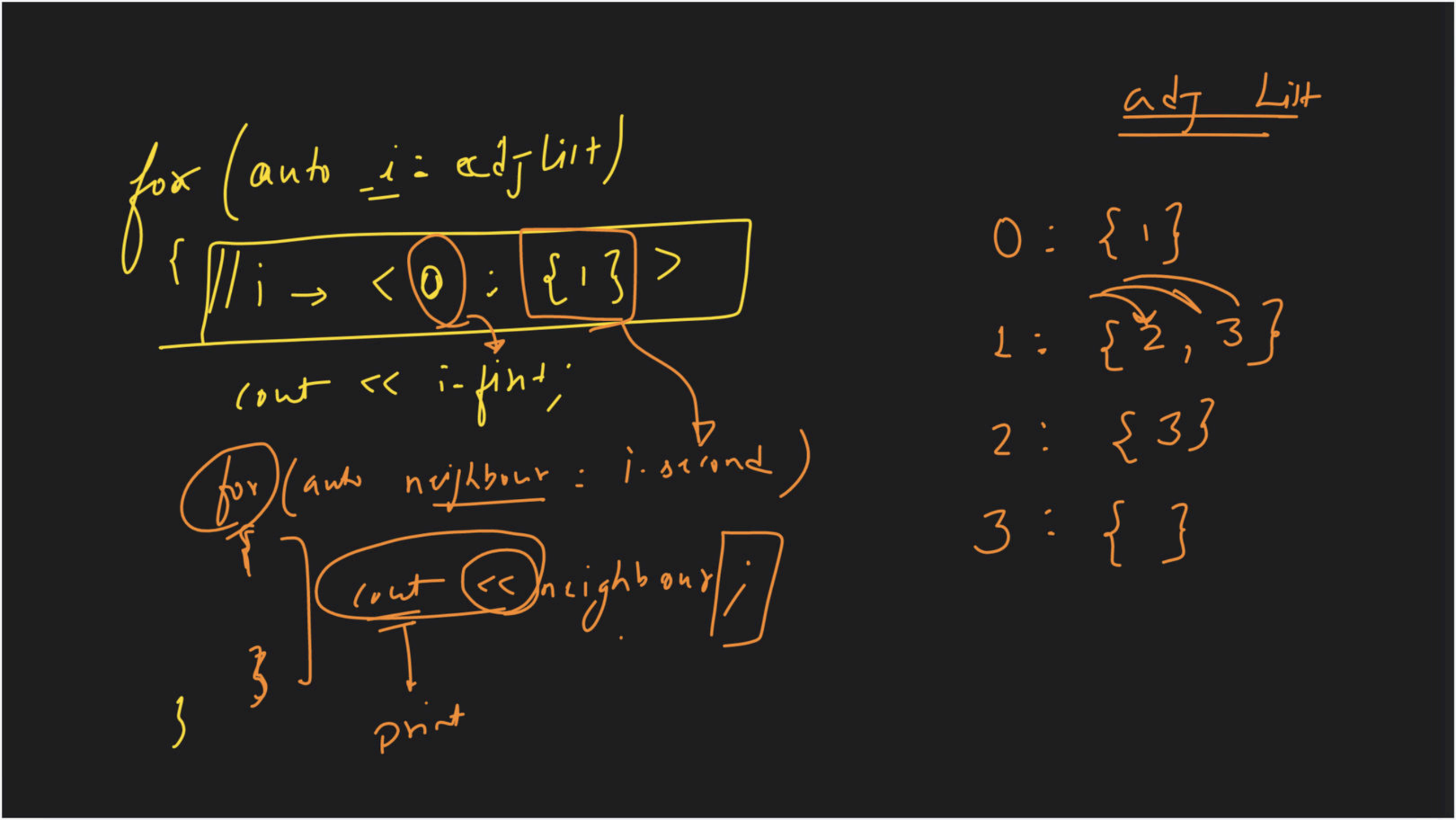

Components
a
b
c
d
e
Connected graph
Component
Component
Component
f
disconnected graph
graph

2 min
Break


① clone a graph
②

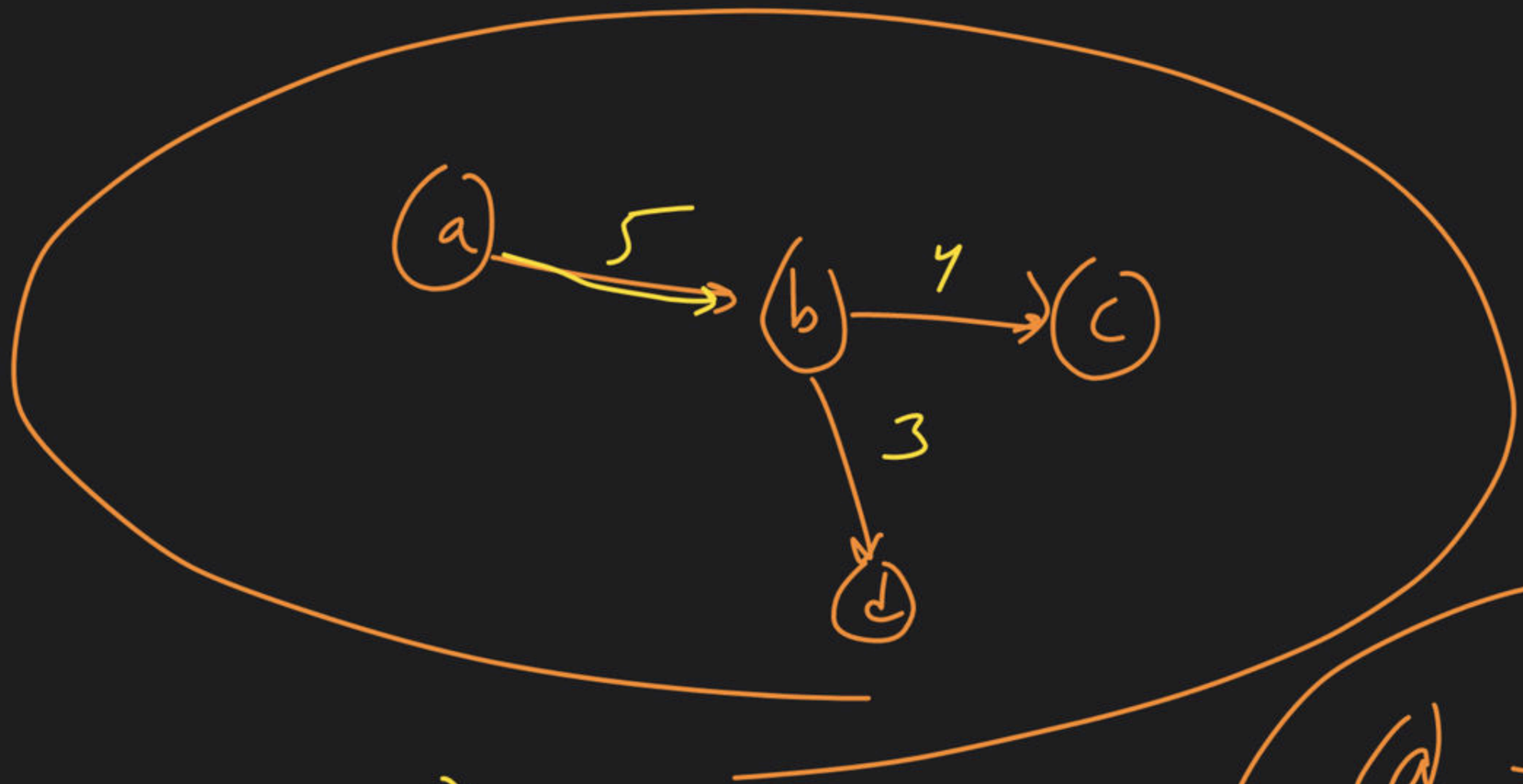

Graph creation
Adjacency matrix
Adjacency List
map <int, list<int>>
Adj List
0
1
2
3
4
5


directin
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

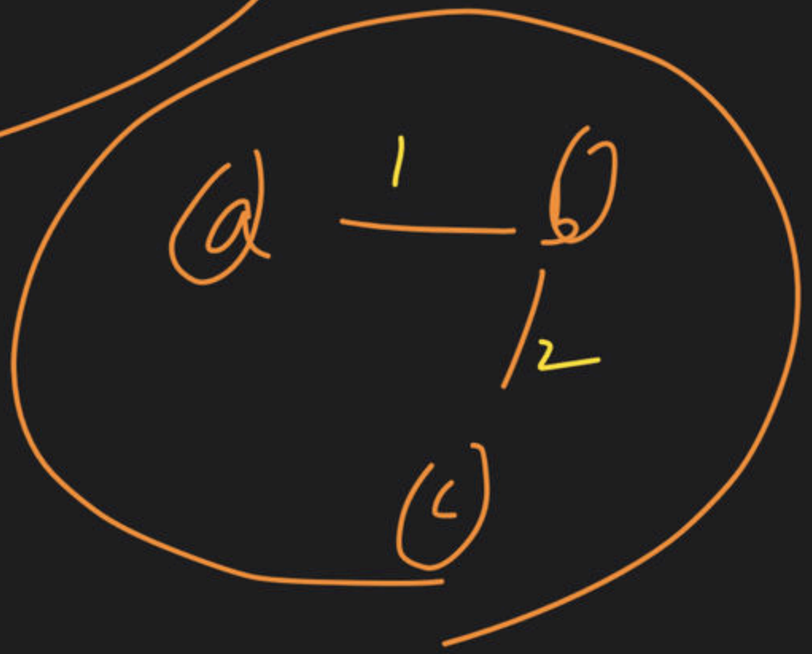
weighted

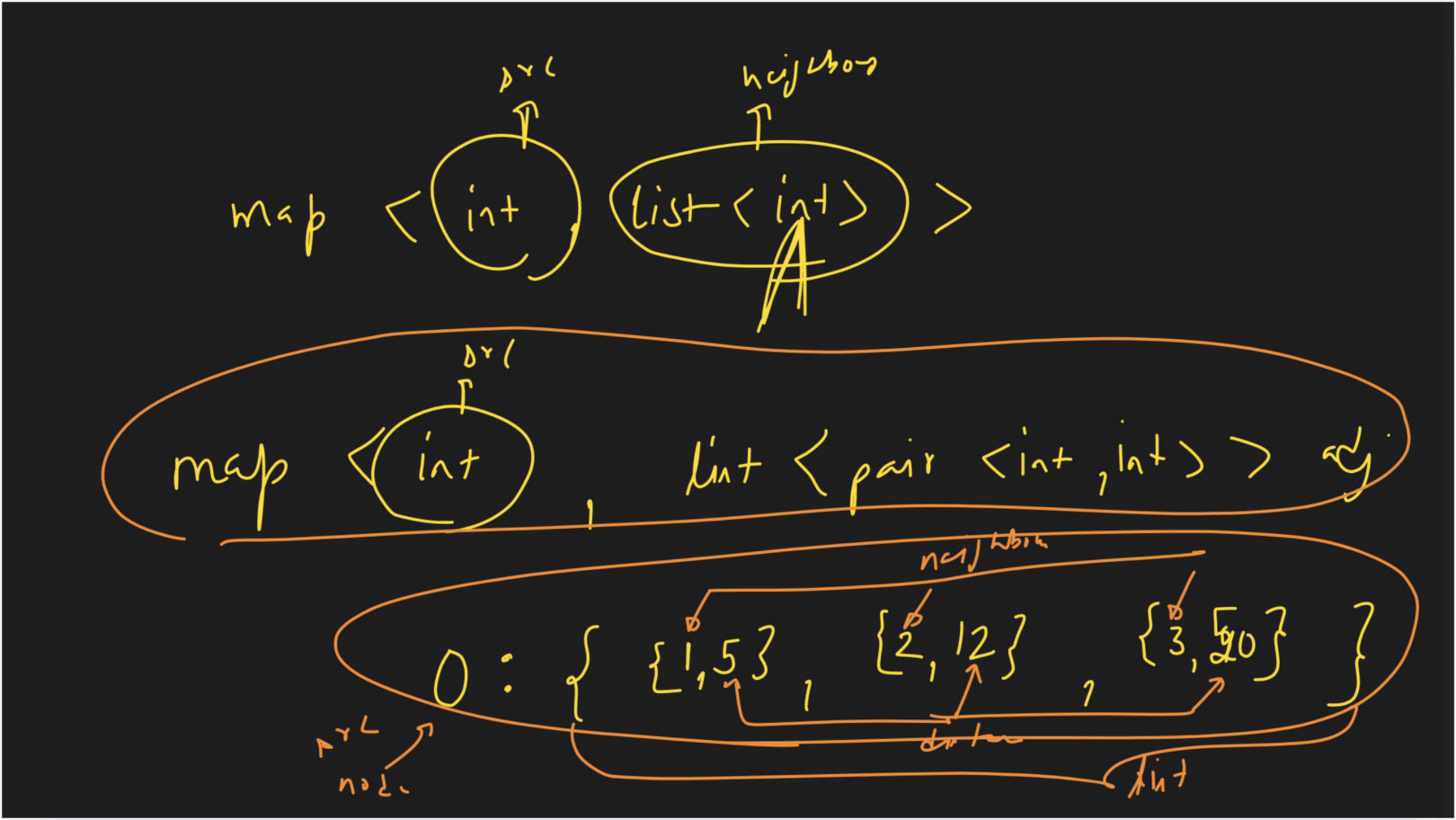


a —1— b

b —7— c

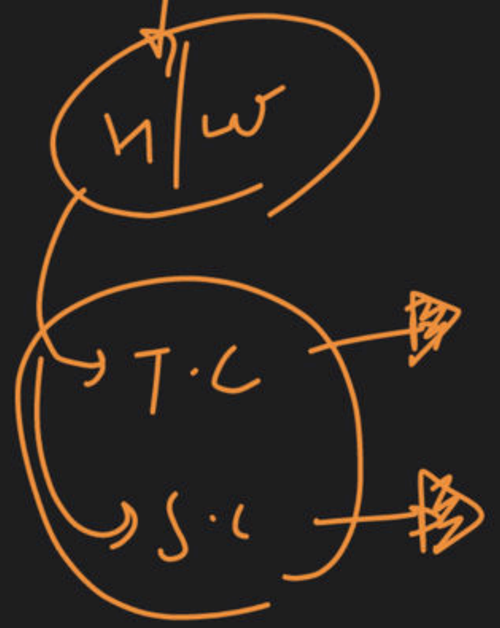
b —3— d



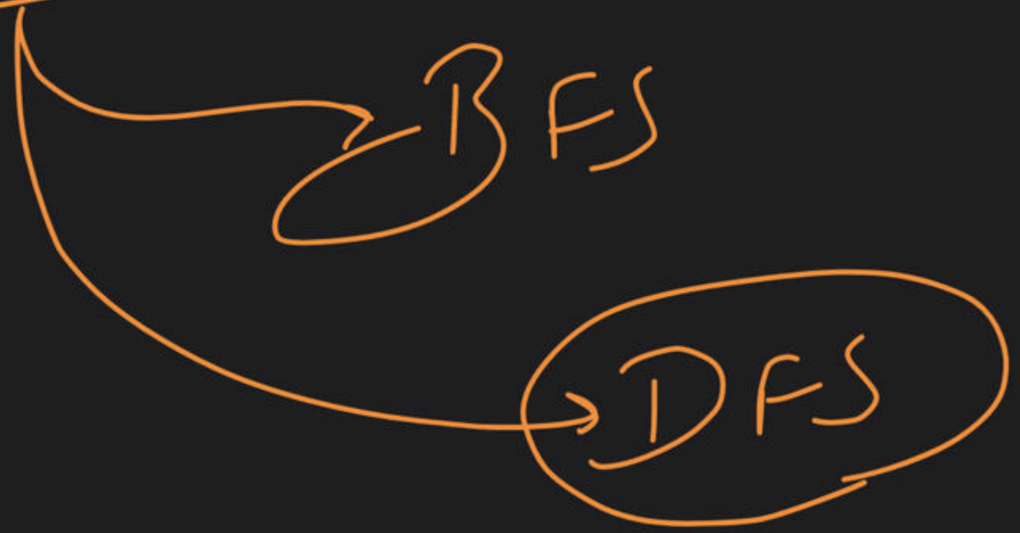

map
int
list<int>
map
int
list<pair<int,int>>
adj:
0:
{1,5}
{2,12}
{3,20}
node

graph

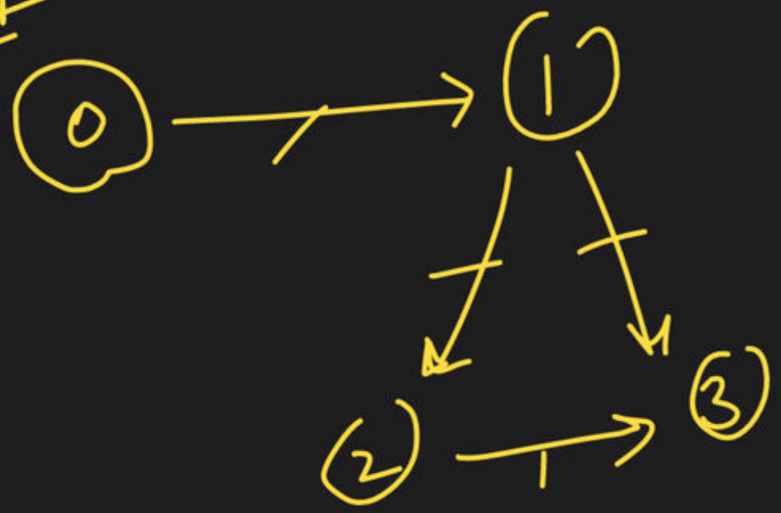
adj List



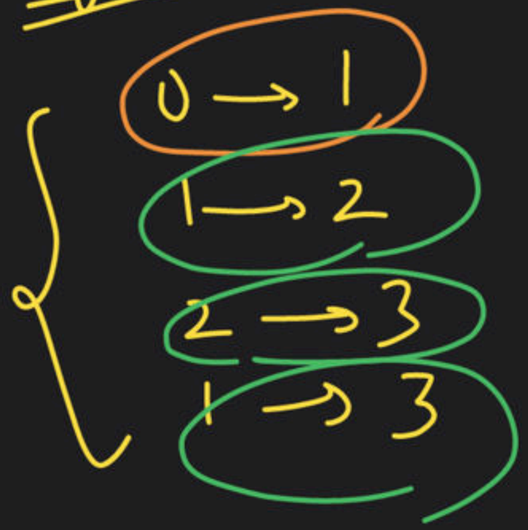
Graph Traversal



graph



edge list

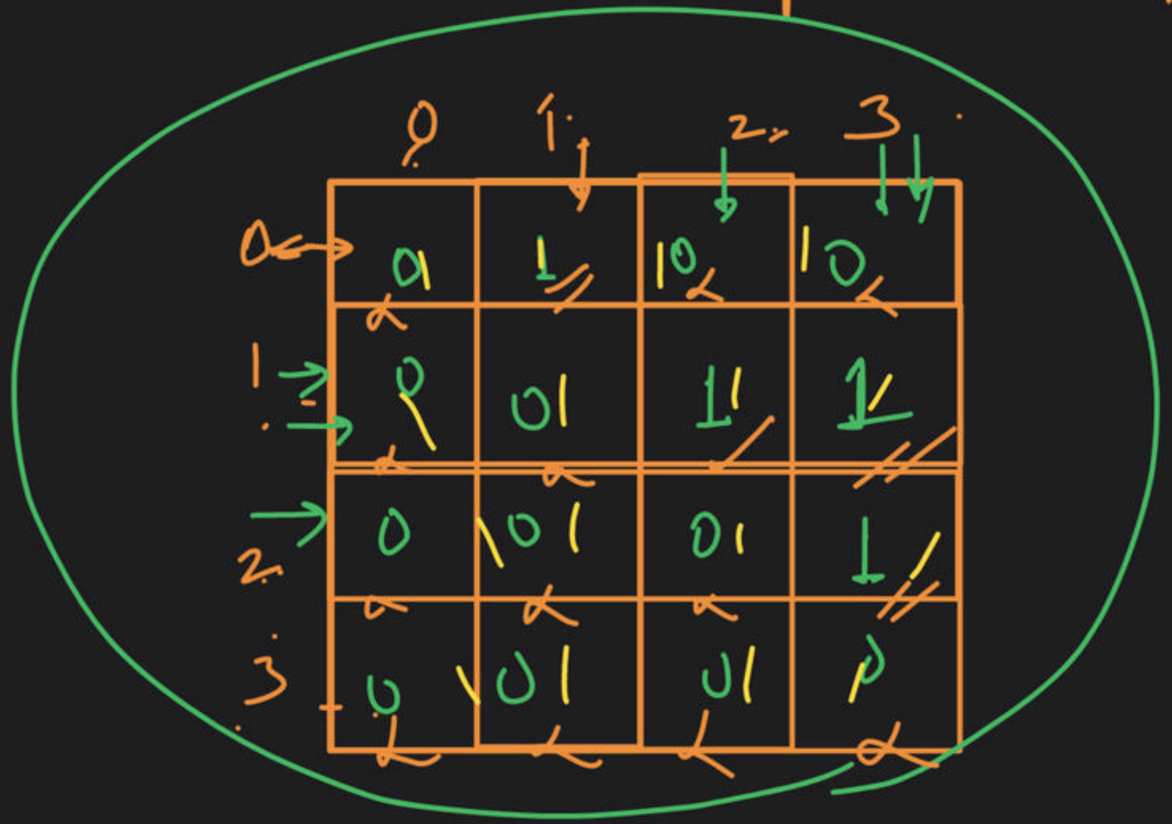


Adjacency matrix

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;
    }
}
```

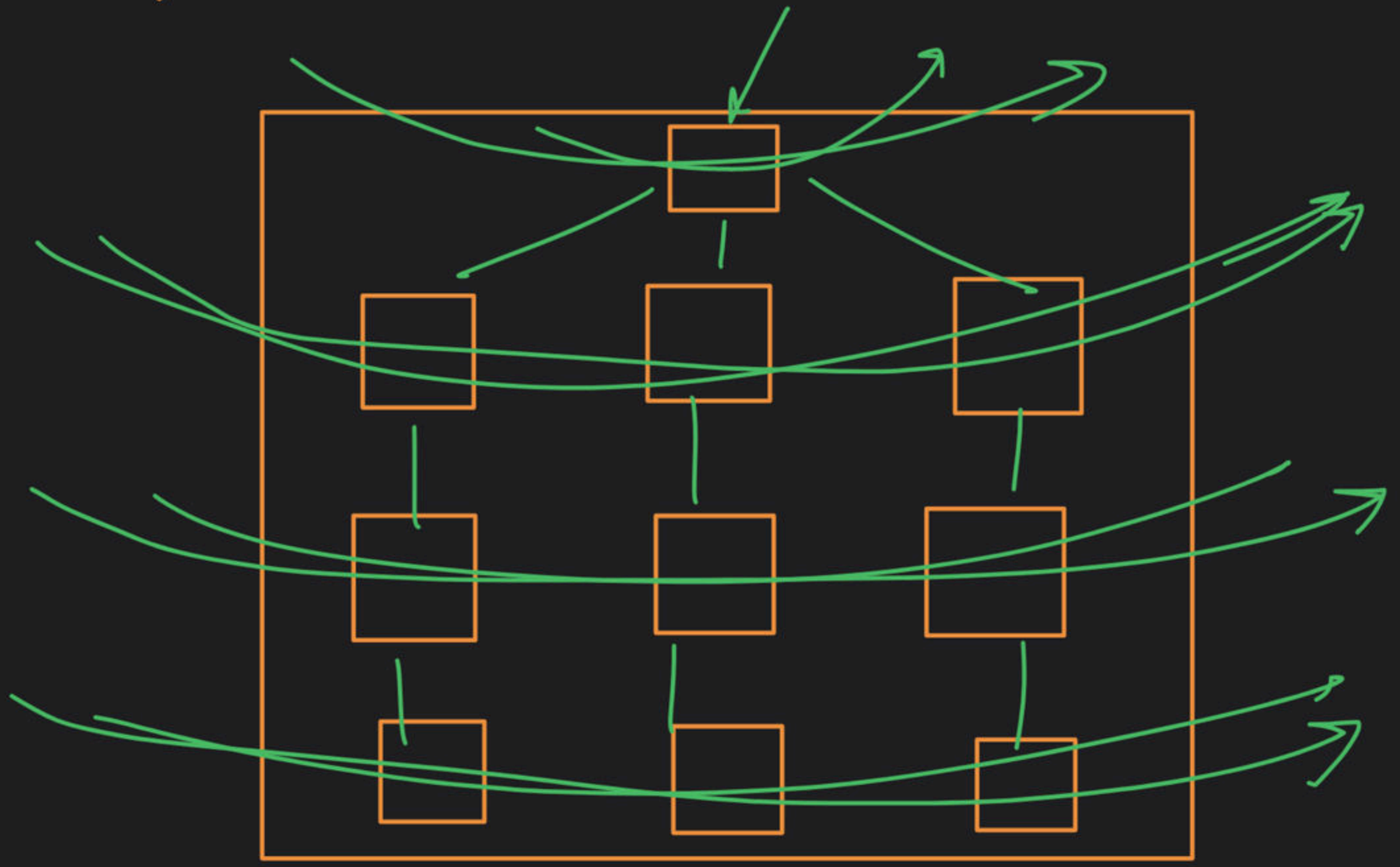
i → pair <int, int>

u → v

S.C → $O(n^2)$

BFS :- Breadth first Search

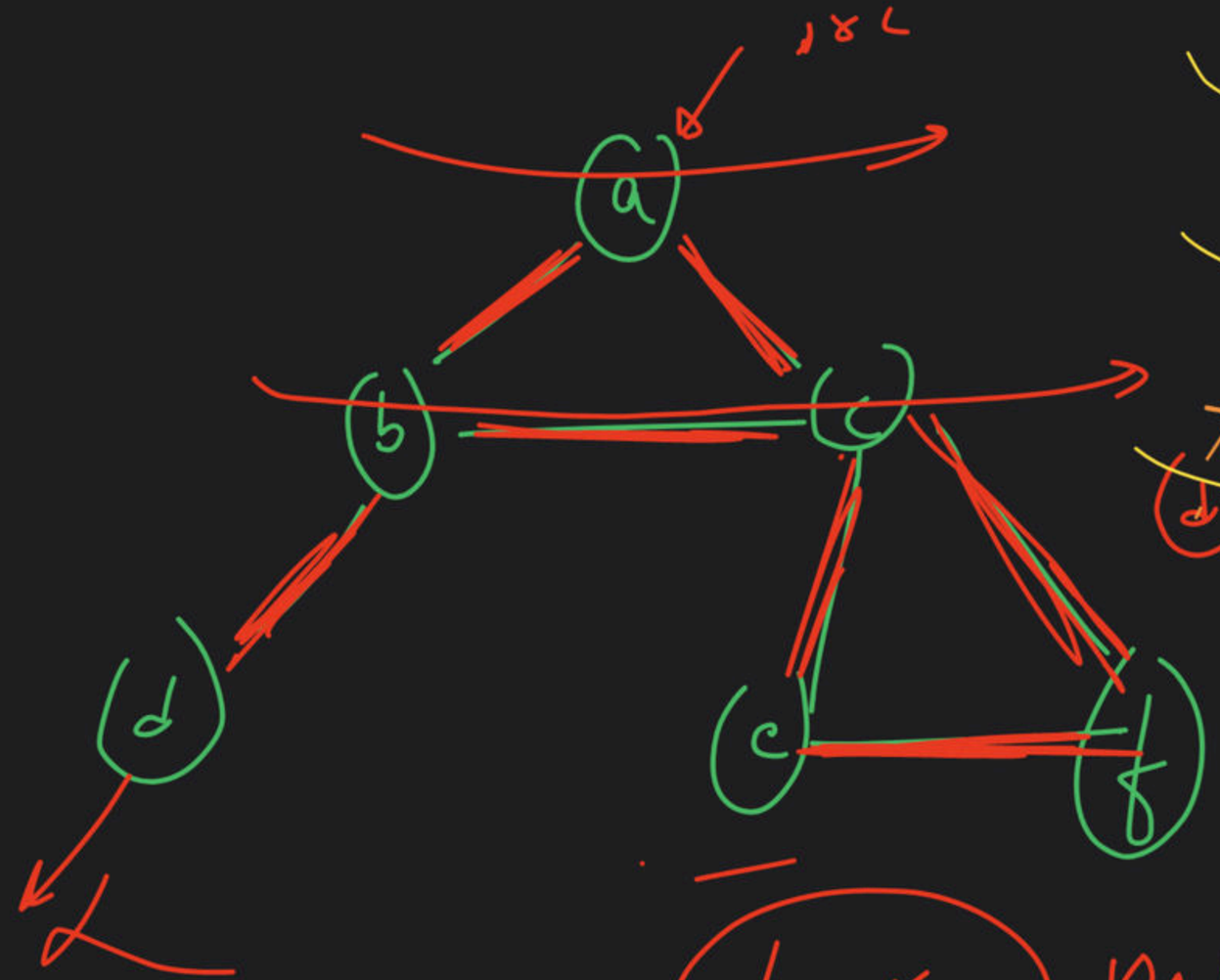
queue

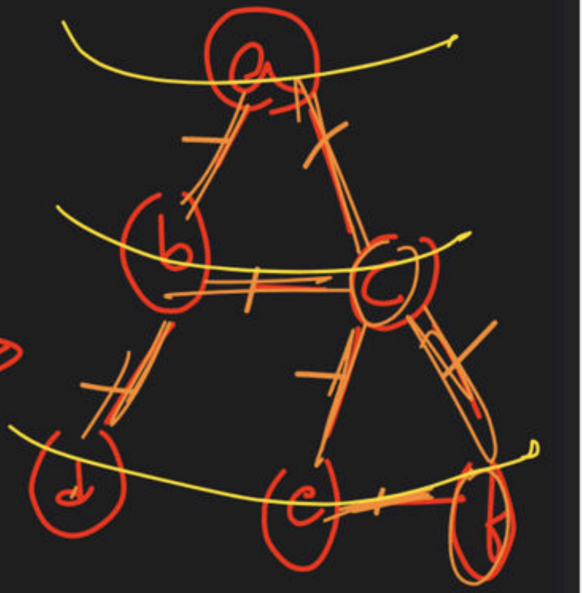


a

b c

d e f





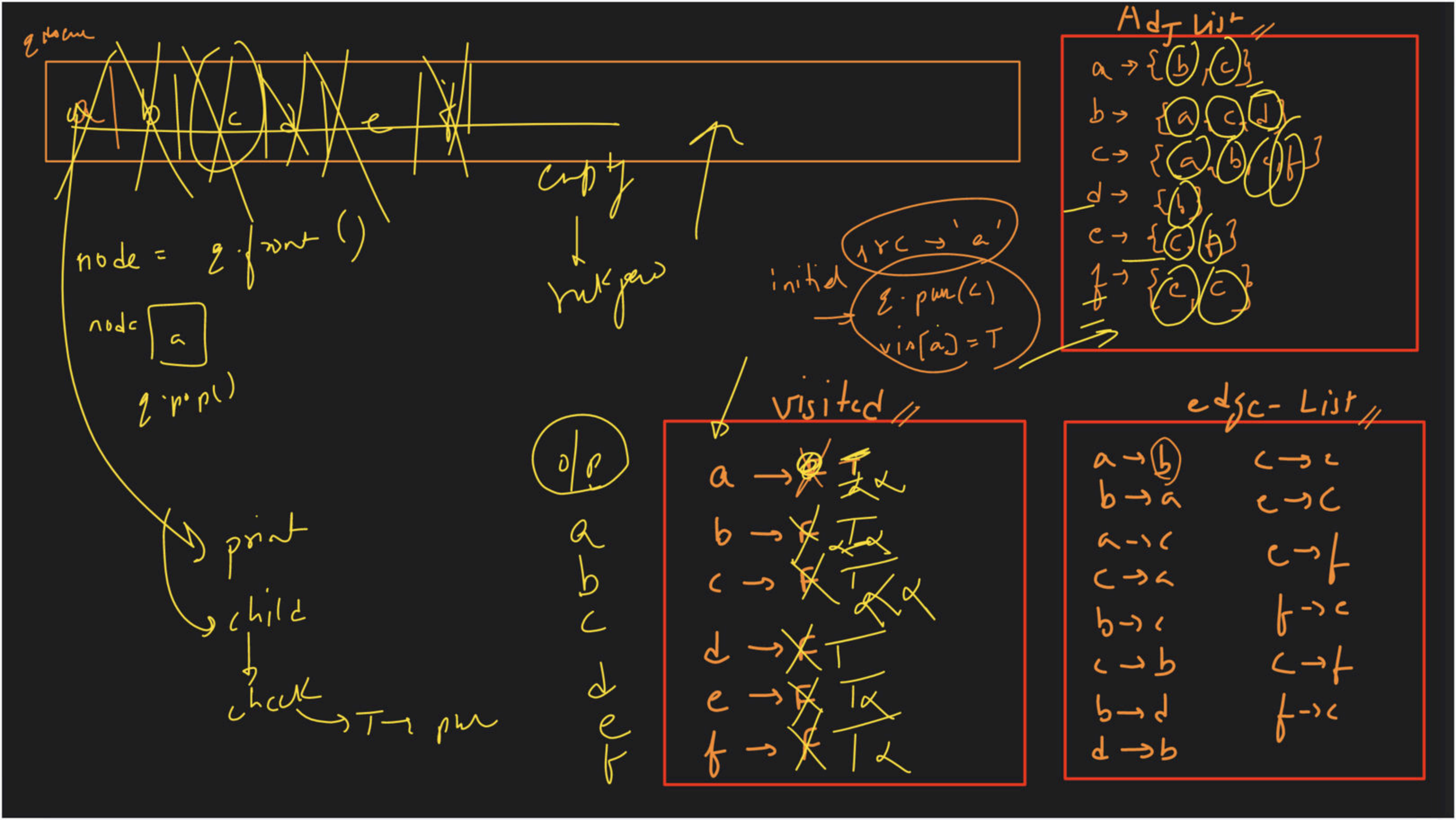
track

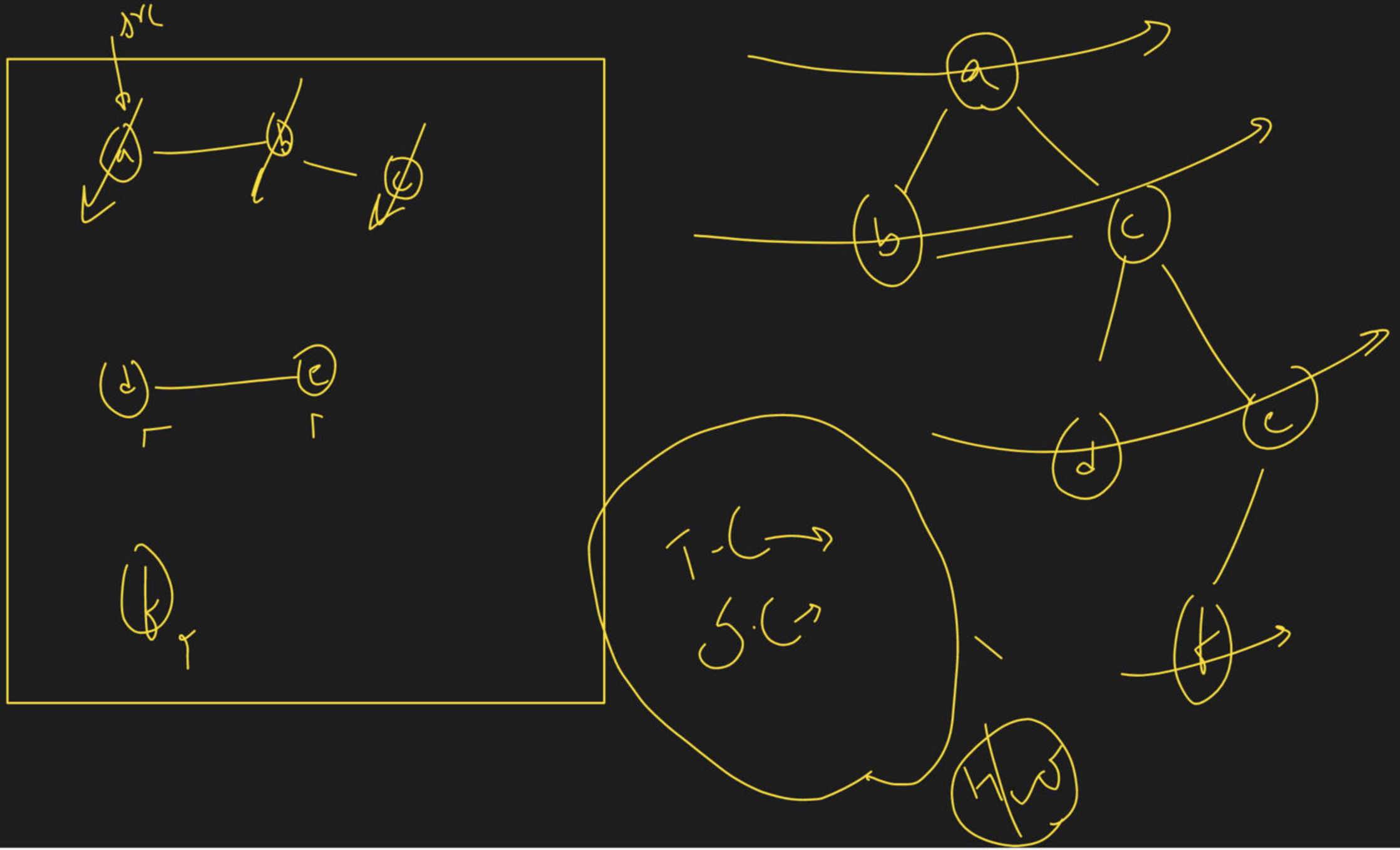
node

visited

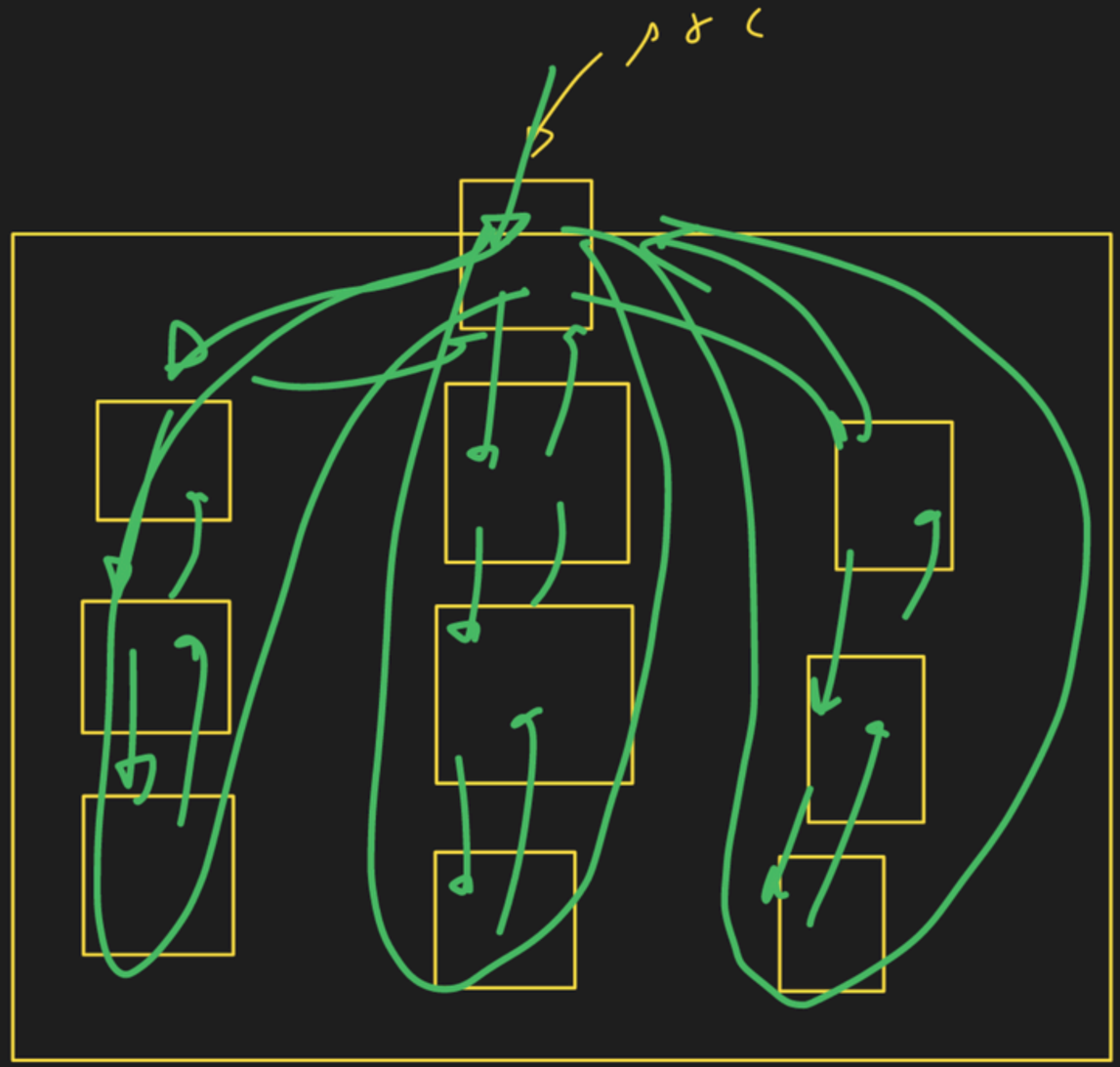
! visited

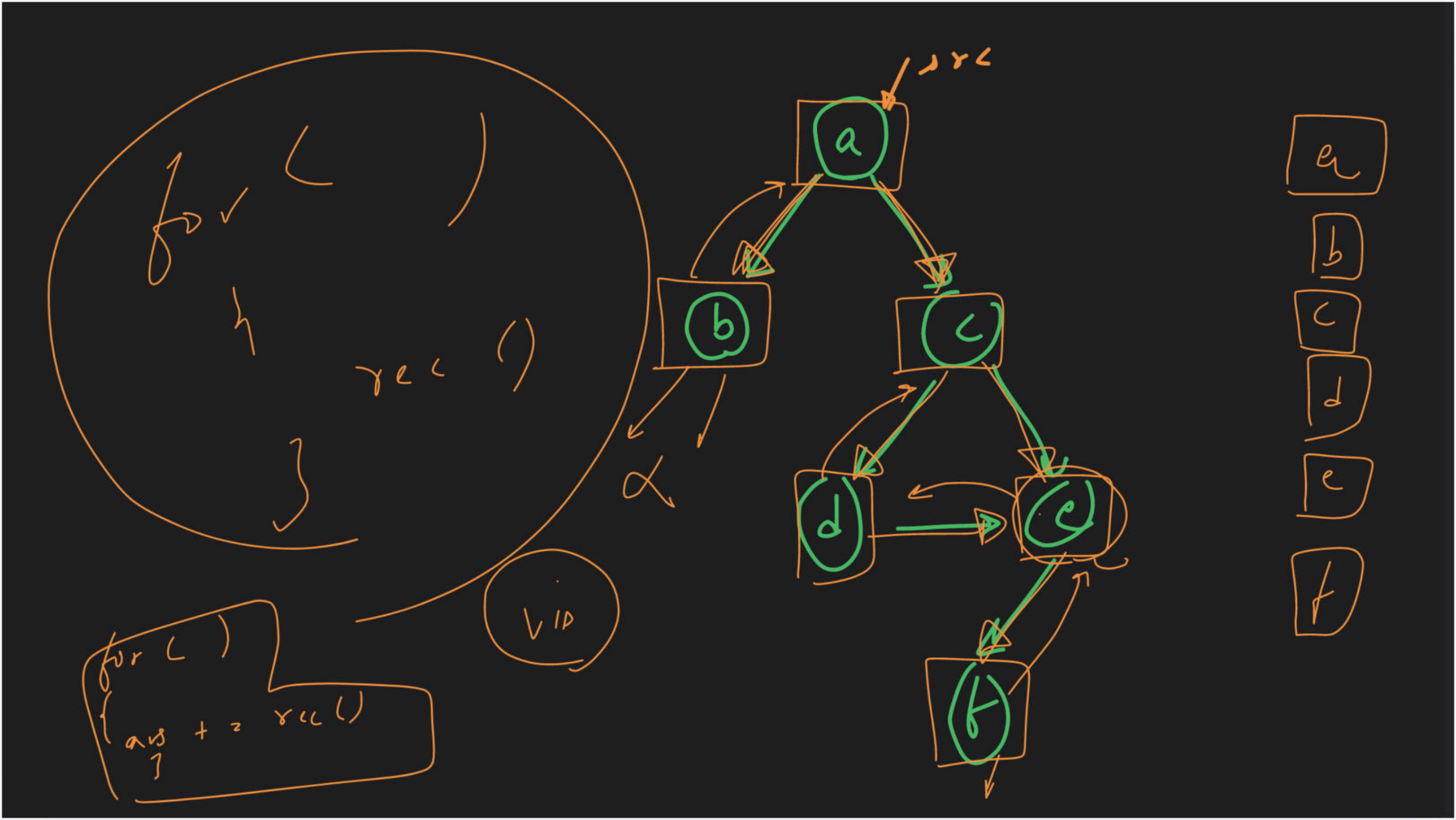
D.S → T/F

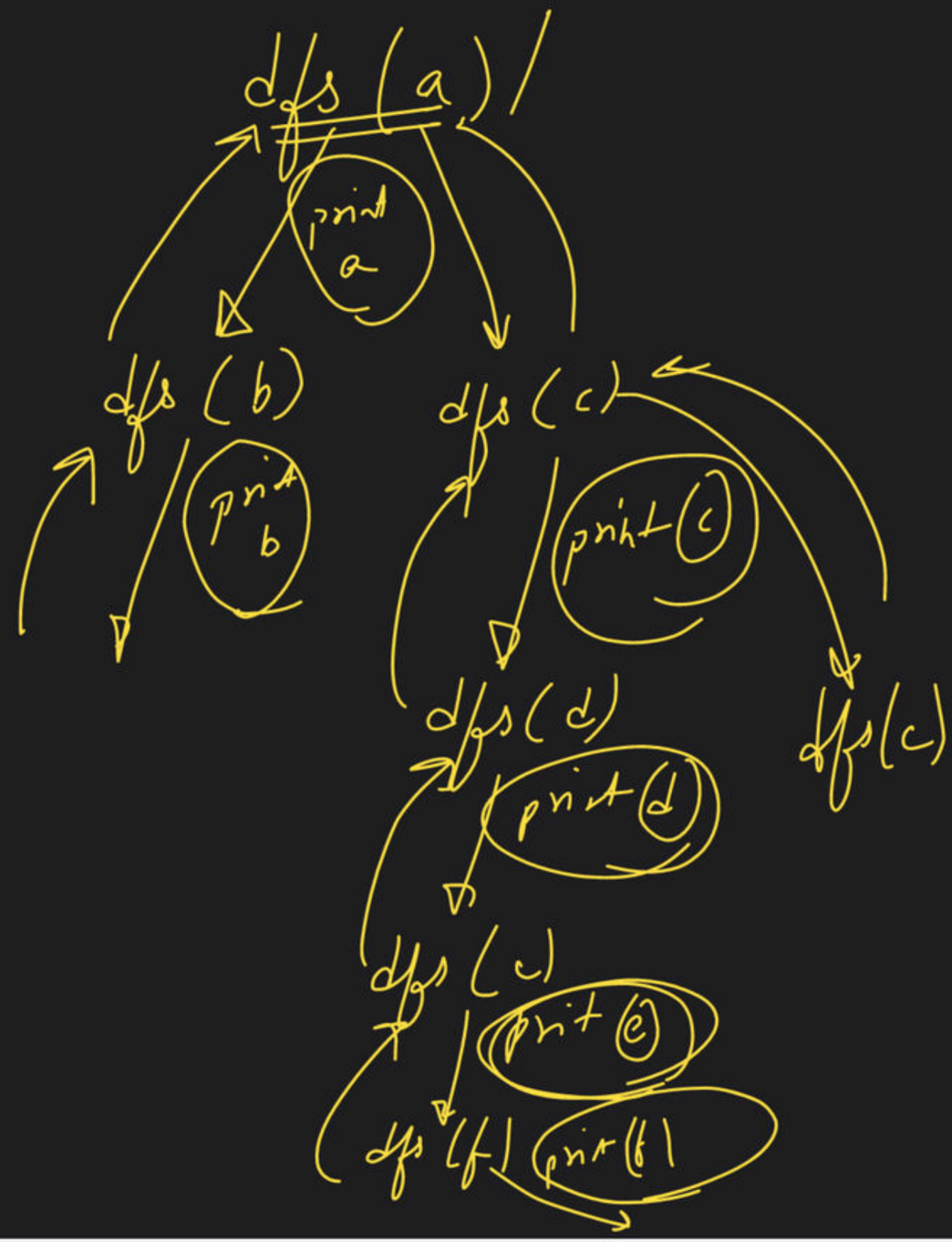

queue
a
b
c
e
f
node = q.front()
node
a
q.pop()
print
child
check
T → push
empty
mark as
visited
src → 'a'
initial
q.push(a)
vis[a] = T
Adj List
a → {b, c}
b → {a, c, d}
c → {a, b, e, f}
d → {b}
e → {c, f}
f → {c, e}
o/p
a
b
c
d
e
f
visited
a → T
b → T
c → T
d → T
e → T
f → T
edge - List
a → b
b → a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



DFS




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



a → {b, c}

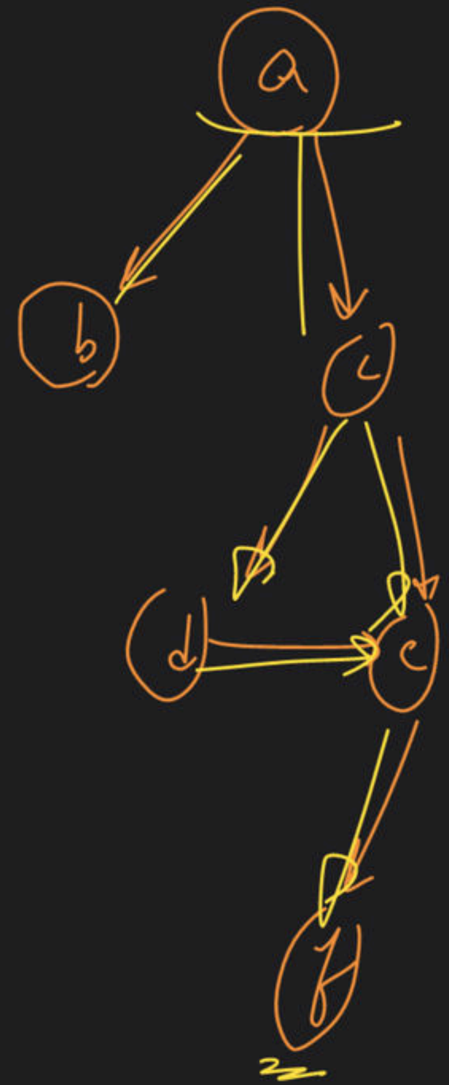
b → {}

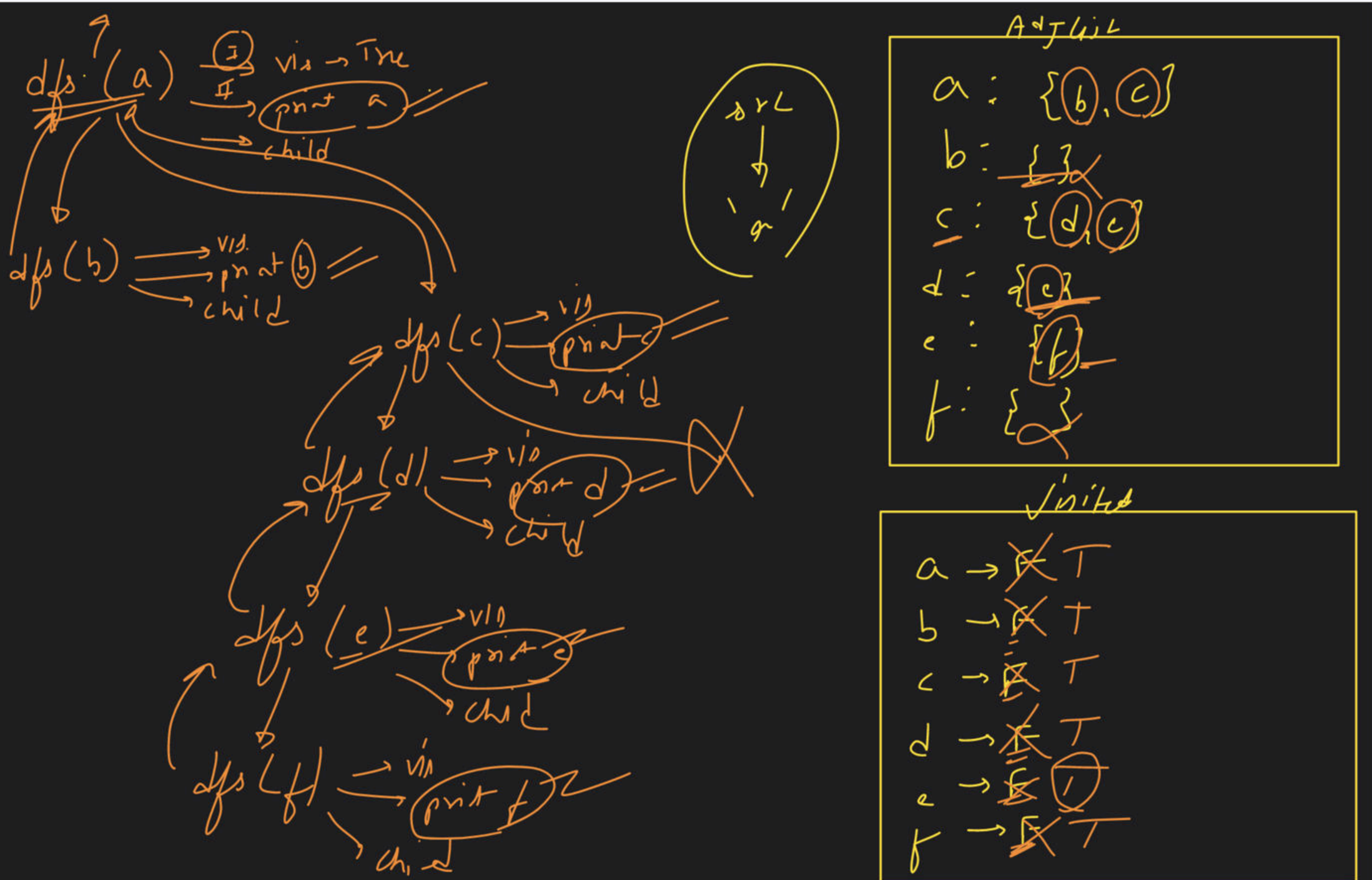
c → {d, e}

d → {e}

e → {f}

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
Adjlist
a : {b, c}
b : {}
c : {d, c}
d : {c}
e : {f}
f : {}
visited
a → F T
b → F T
c → F T
d → F T
e → F T
f → F T