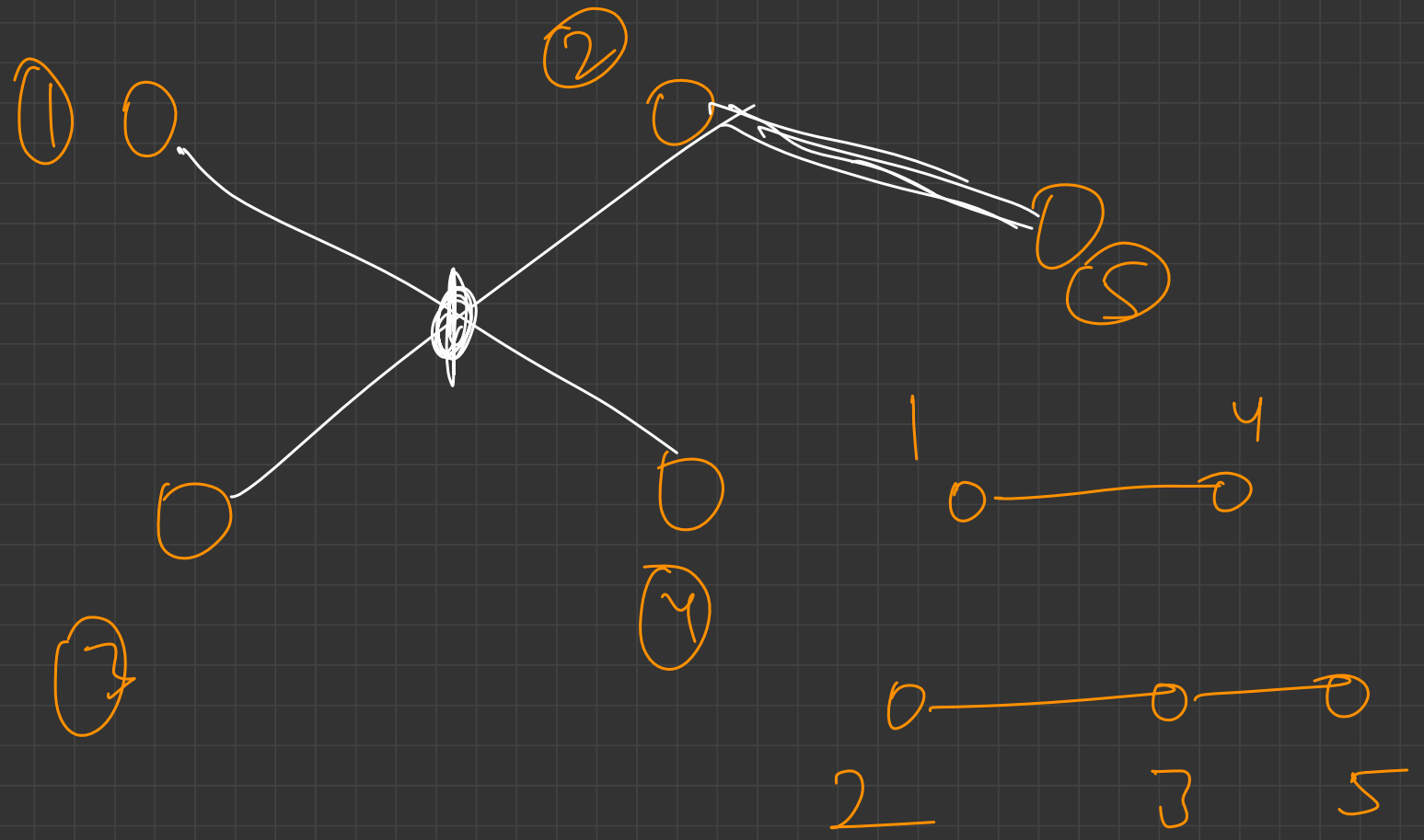
Graphs Class 1

Priyansh Agarwal


1
2
5
4
7
6
1
4
2
3
5

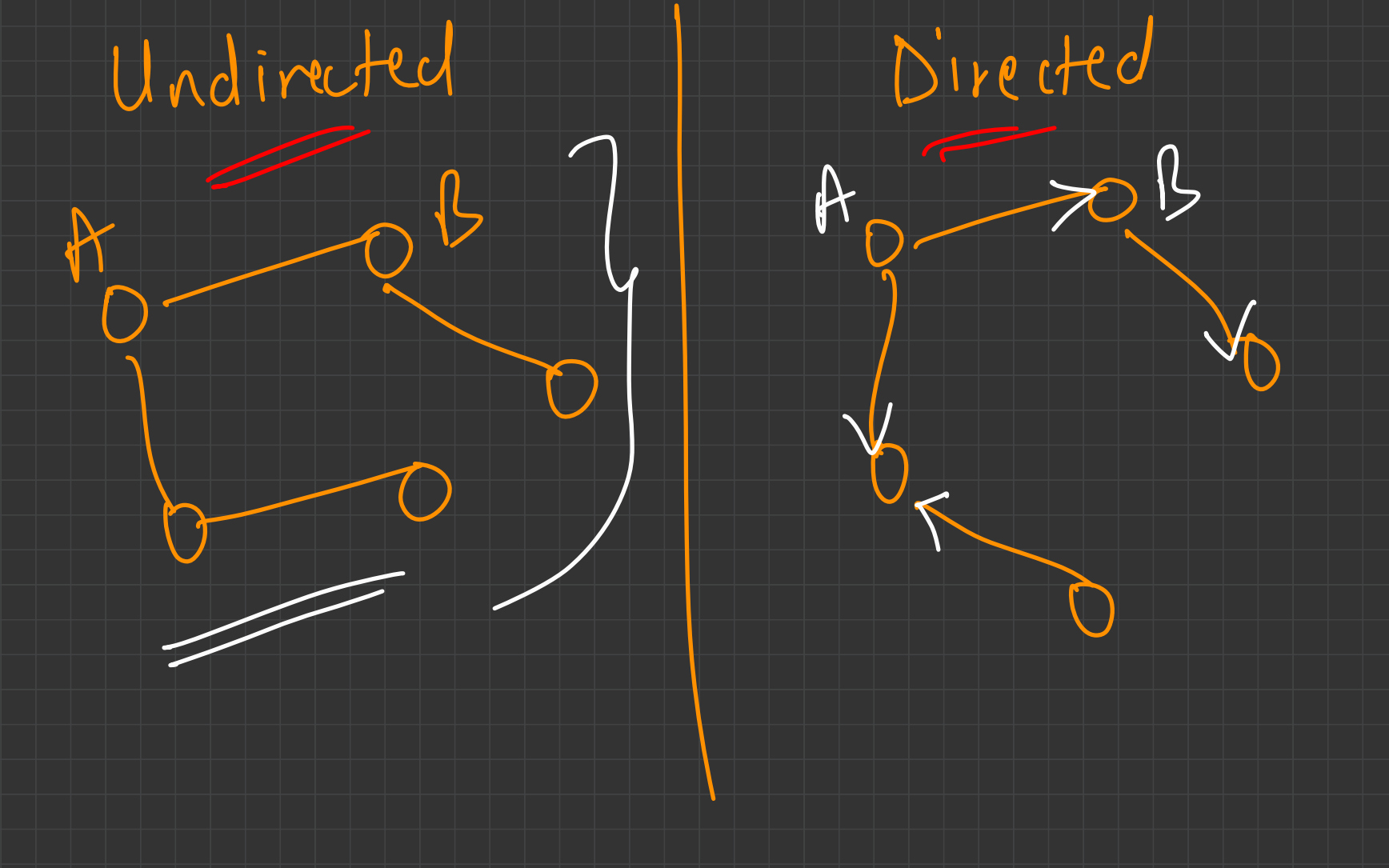
Graph Collection of nodes connected through edges

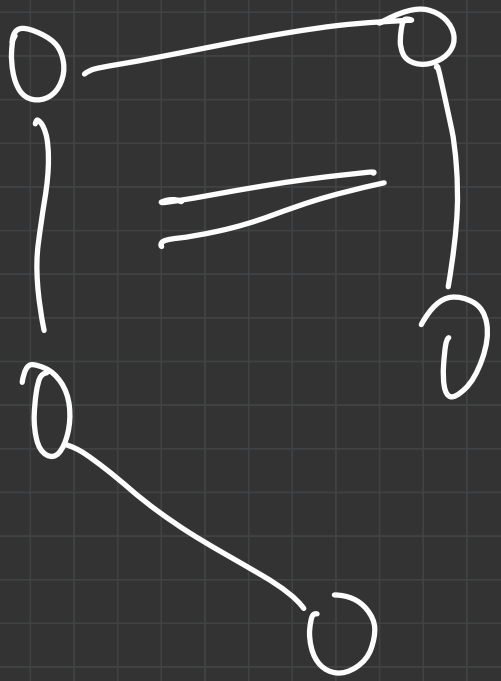
Types of Graphs

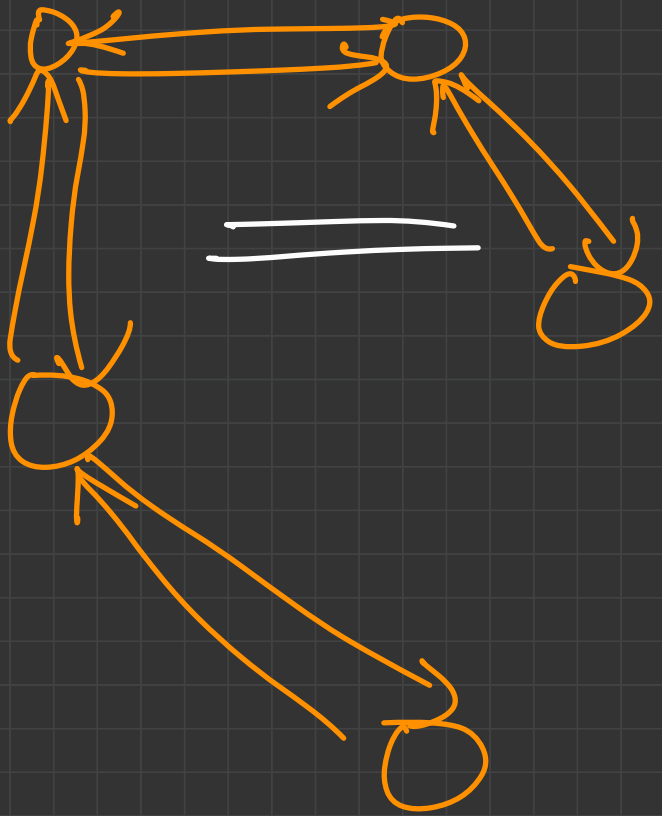
- Undirected vs Directed
- Unweighted vs Weighted
- Cyclic + Acyclic
- Connected + Disconnected
- Complete graph

no. of edges

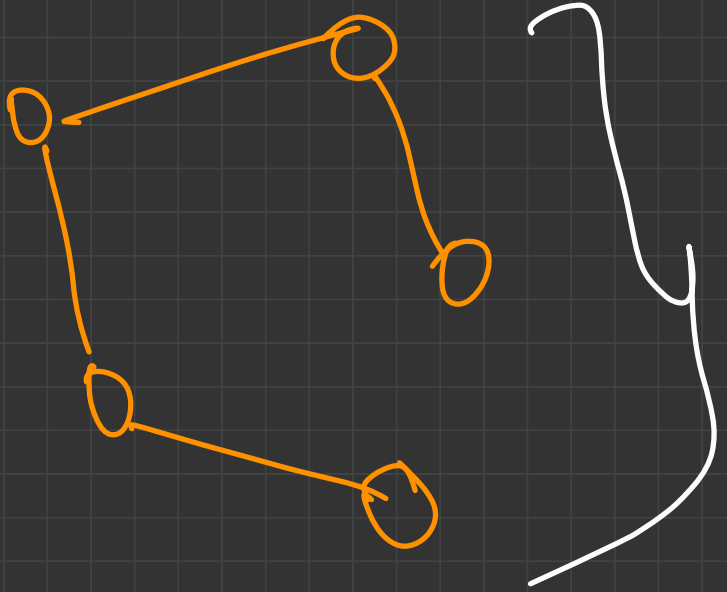
$= (n \cdot (n-1))/2$


Undirected
A
B
Directed
A
B

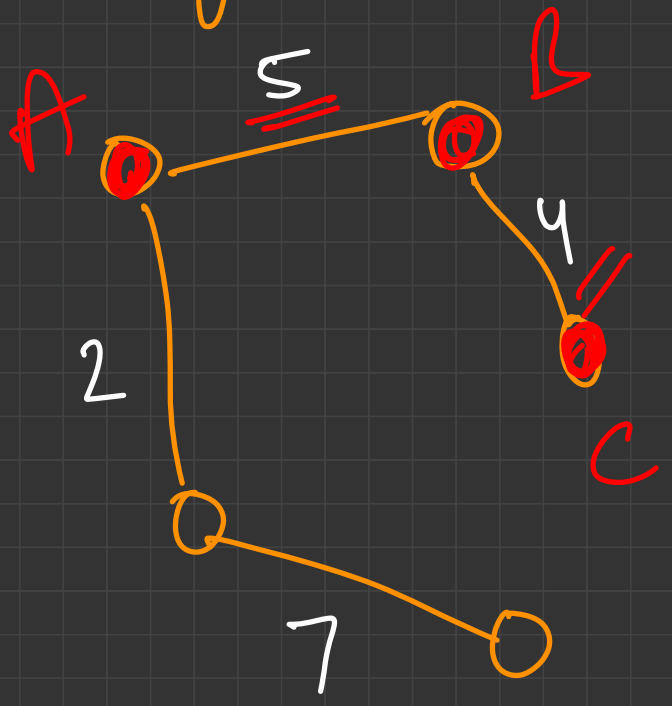




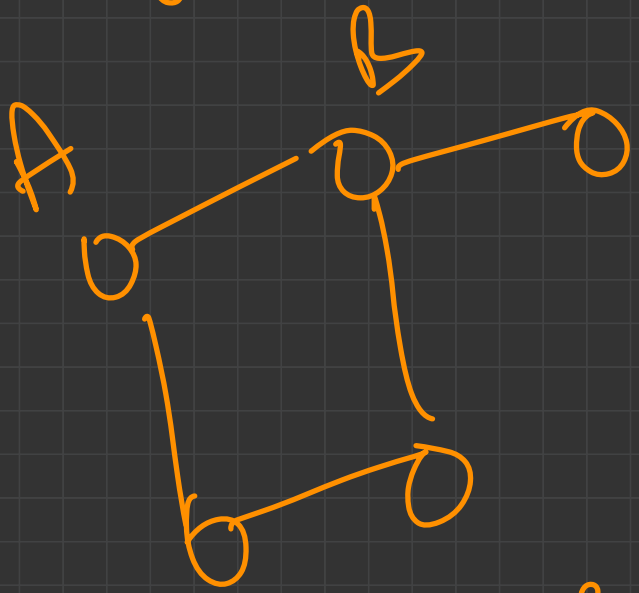
Unweighted



Weighted

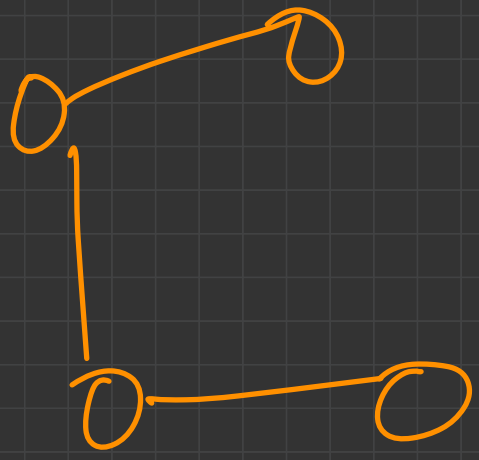


Cyclic

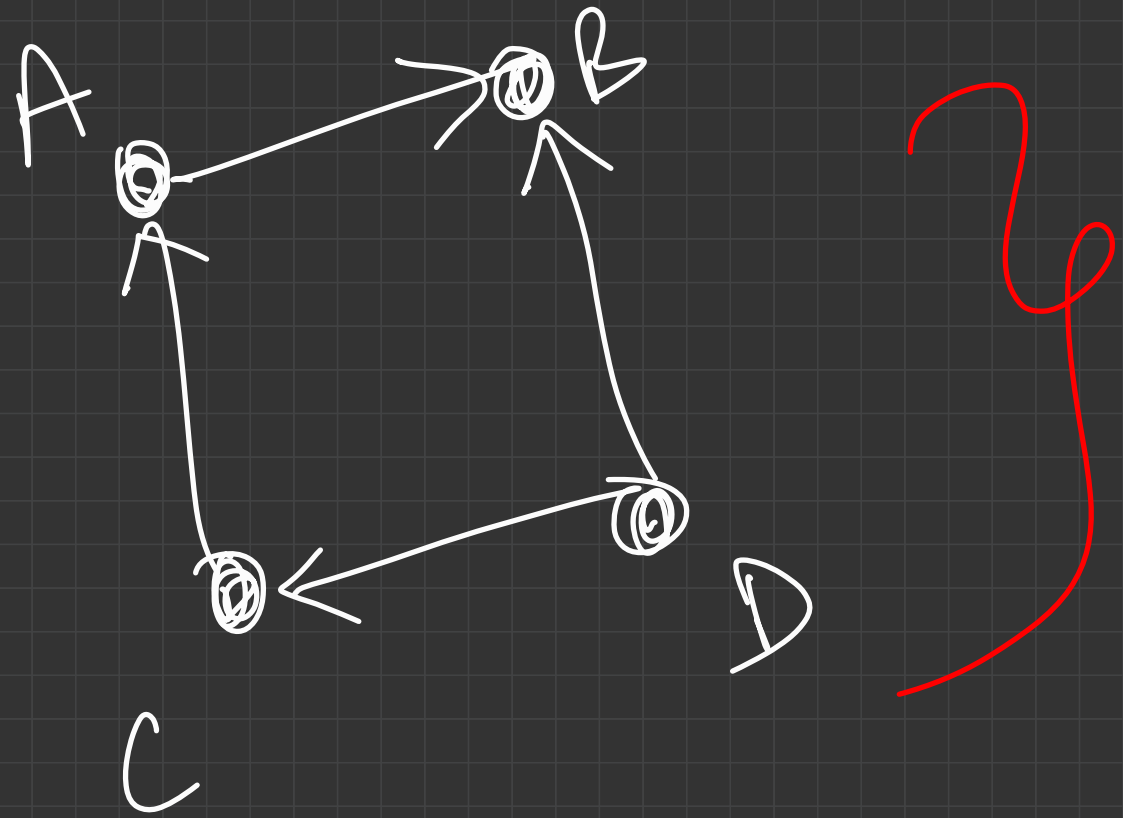


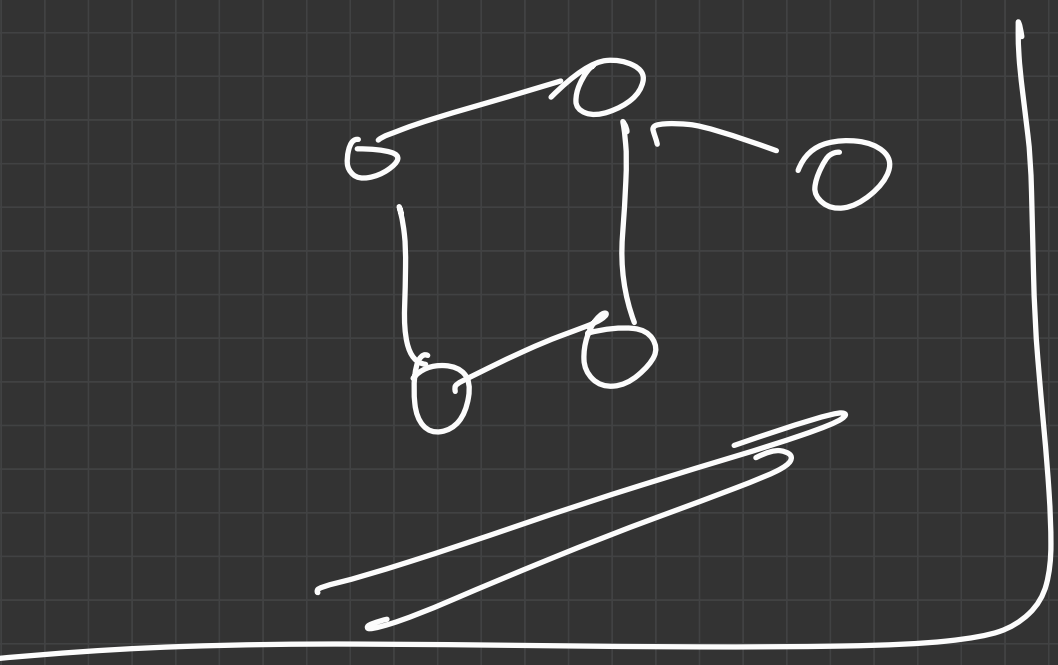
(there exist more than 1 path b/w some 2 nodes

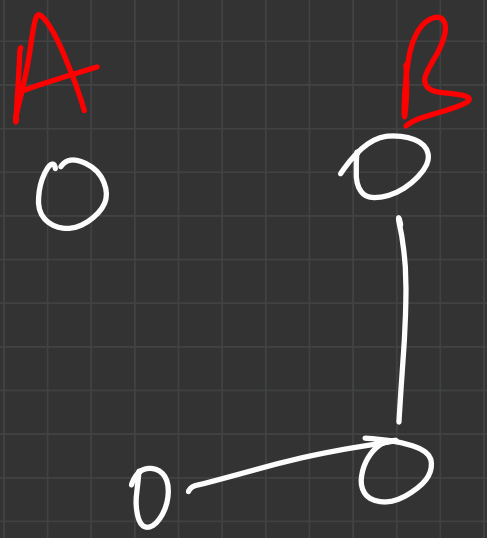
Acyclic



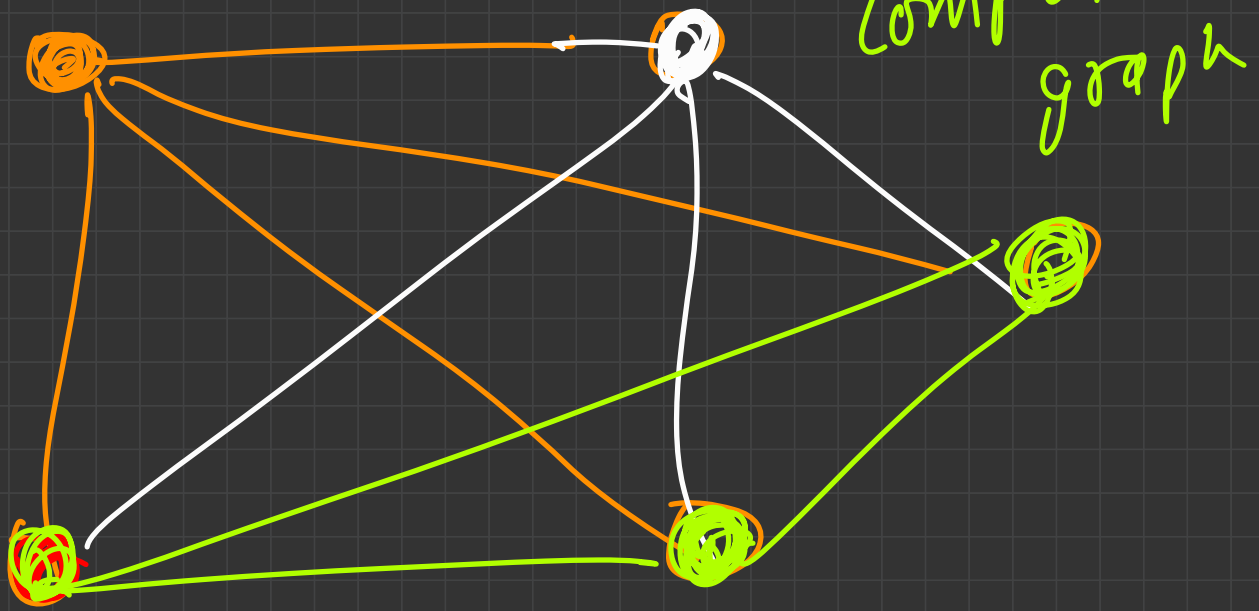
undirected


A
B
C
D





Connected Graph : there exists at least path b/w any 2 nodes

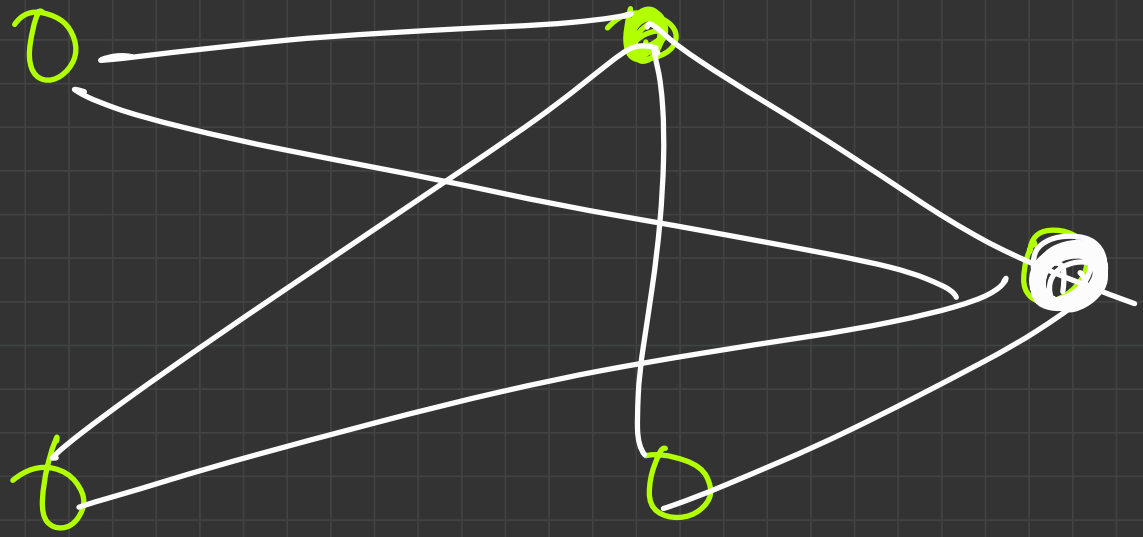

Complete graph

Complete graph : A graph in which there is an edge b/w every pair of nodes

n

nC_2

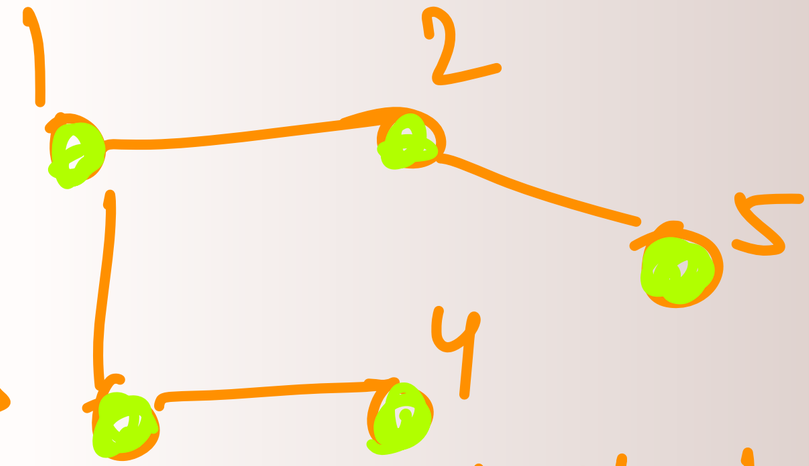
$(n \cdot (n-1))/2$



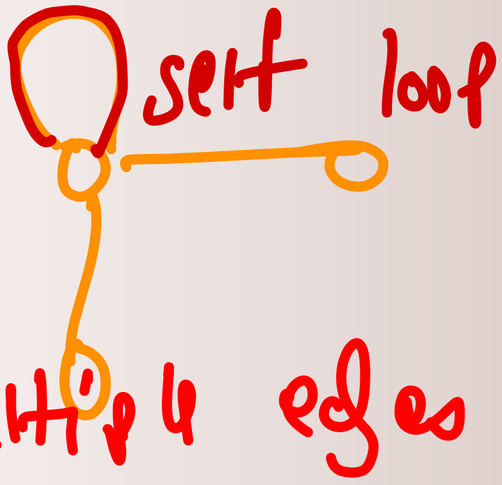
$$\frac{5}{} \qquad \frac{(4+3+2+1)}{(n\cdot(n-1))/2}$$

Common Terms

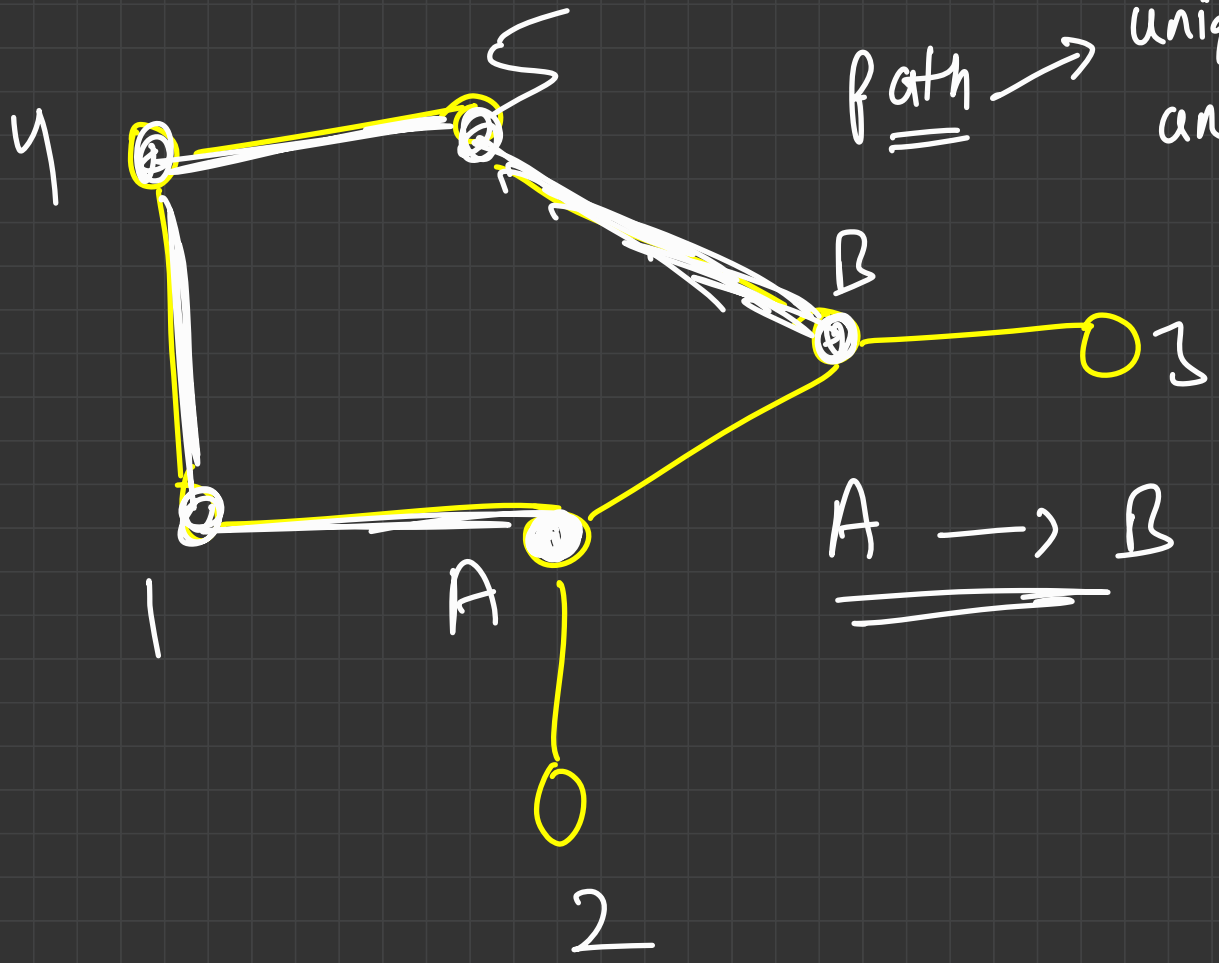
- Vertices + Edges
- Neighbours + Degree
- Self loop
- Path + Walk + Cycle
- Simple Graph
- Bridge + Articulation Point

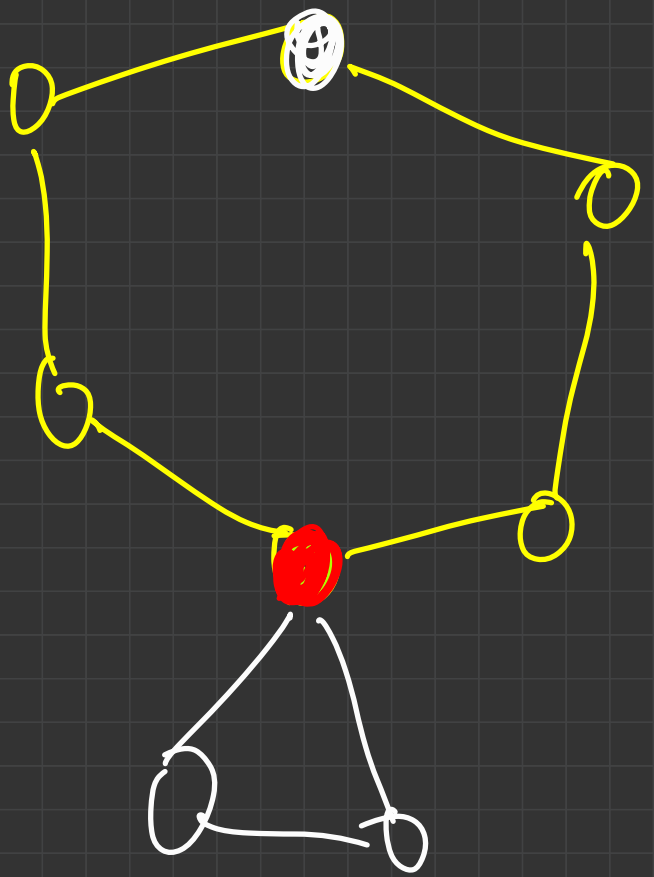


every node reachable by just 1 edge



graph without self loops & multiple edges


path → unique nodes and edges
y
S
B
3
A → B
1
A
2

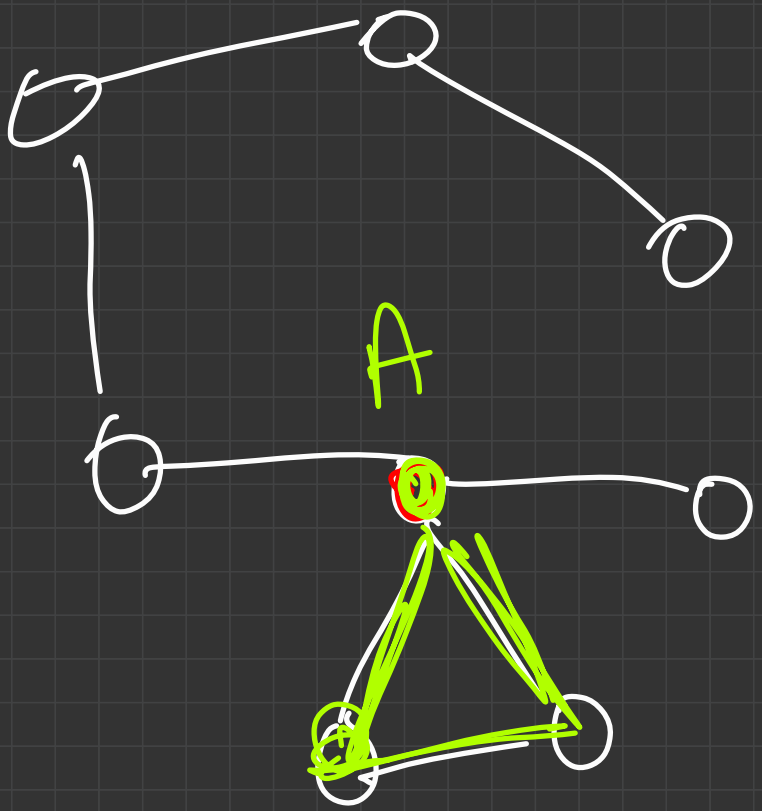


Walk

A -> B

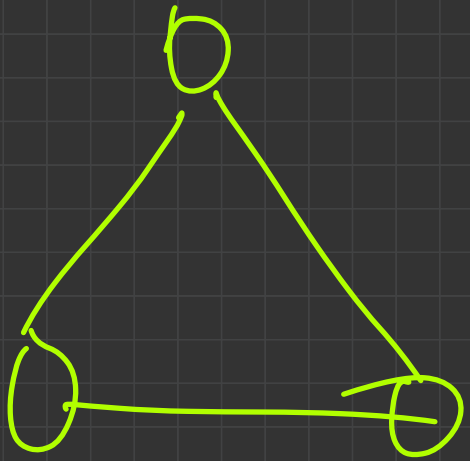
without caring about unique nodes and edges

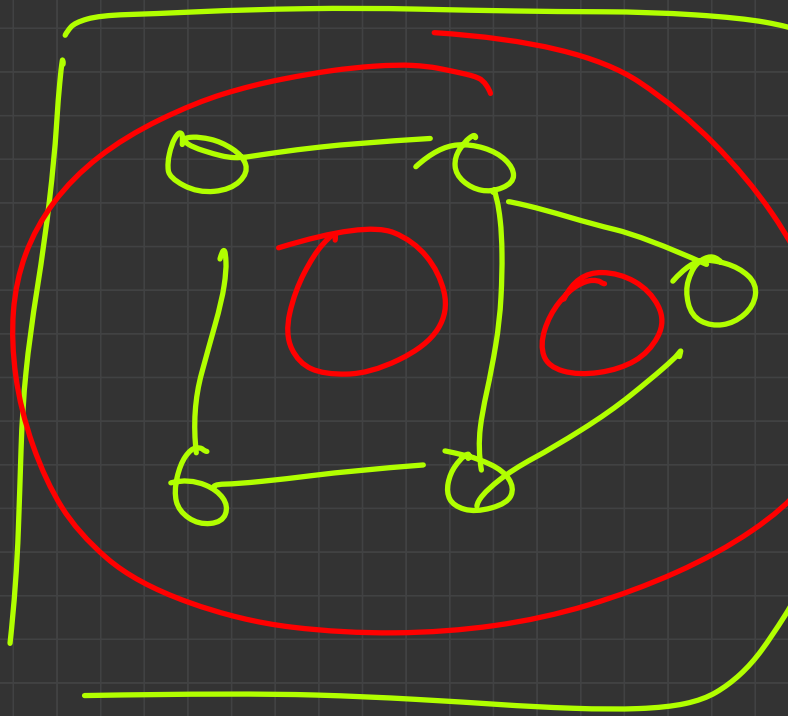
Cycle

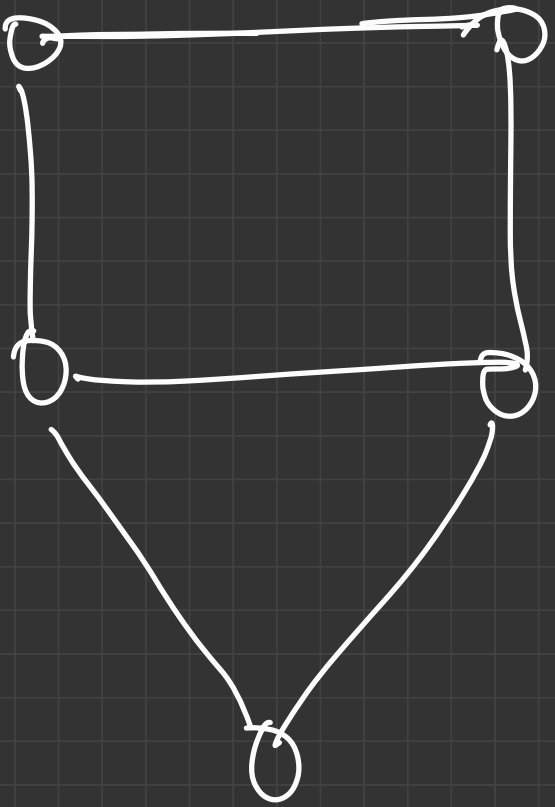


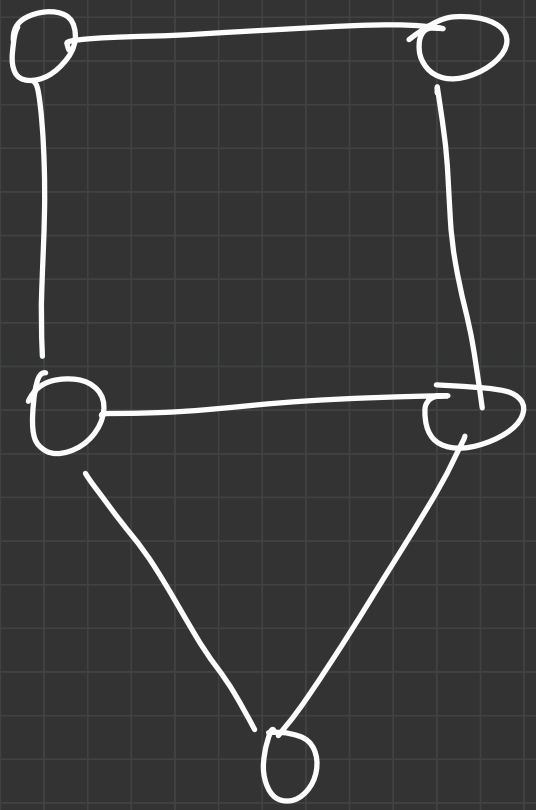
A → A

with unique nodes and edges except node A



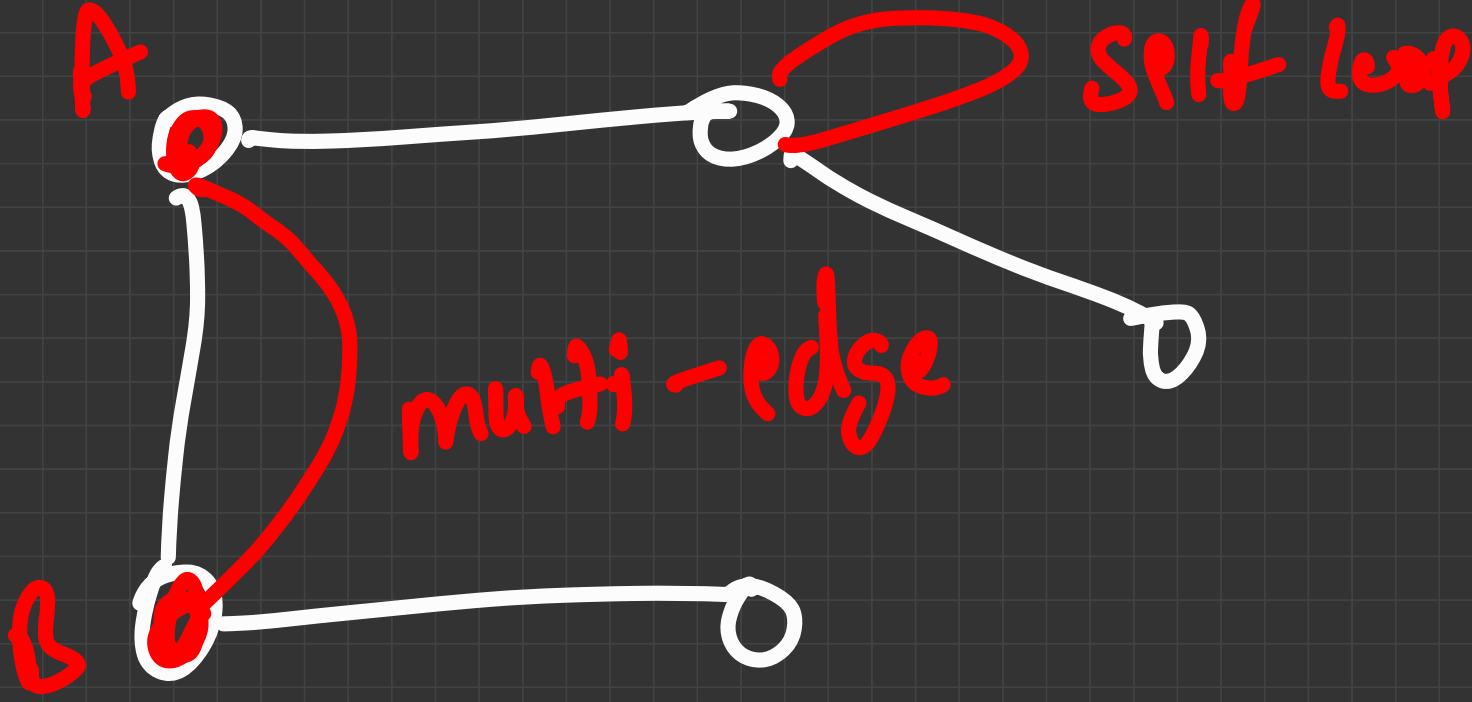


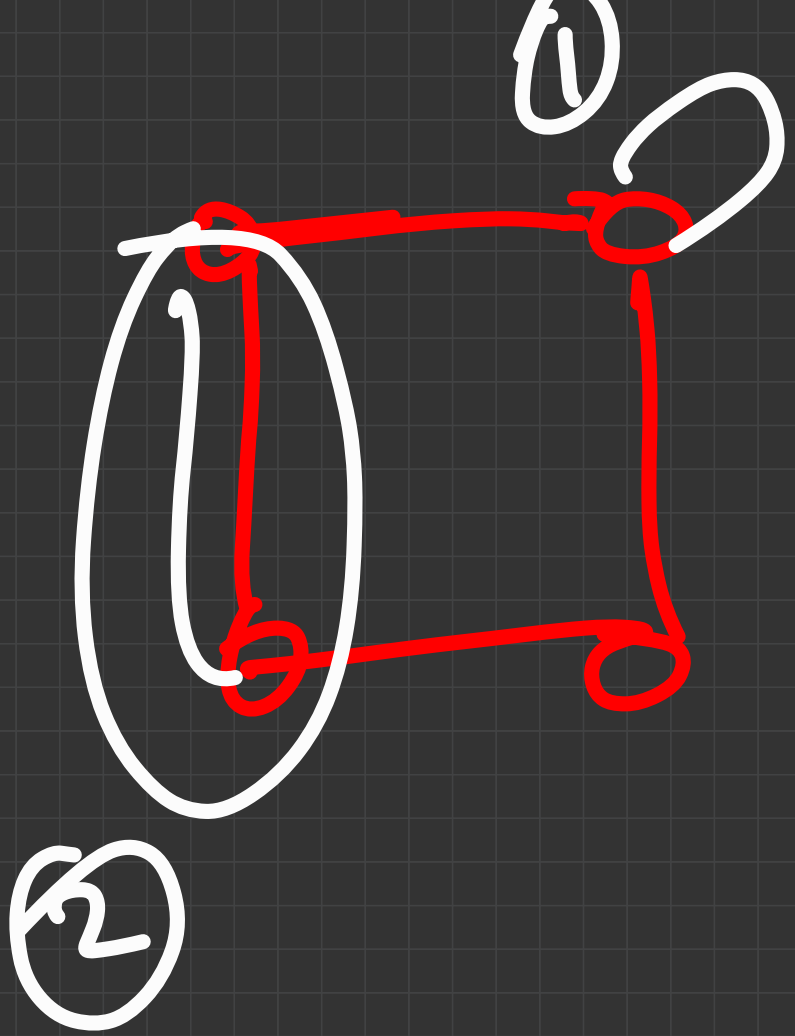




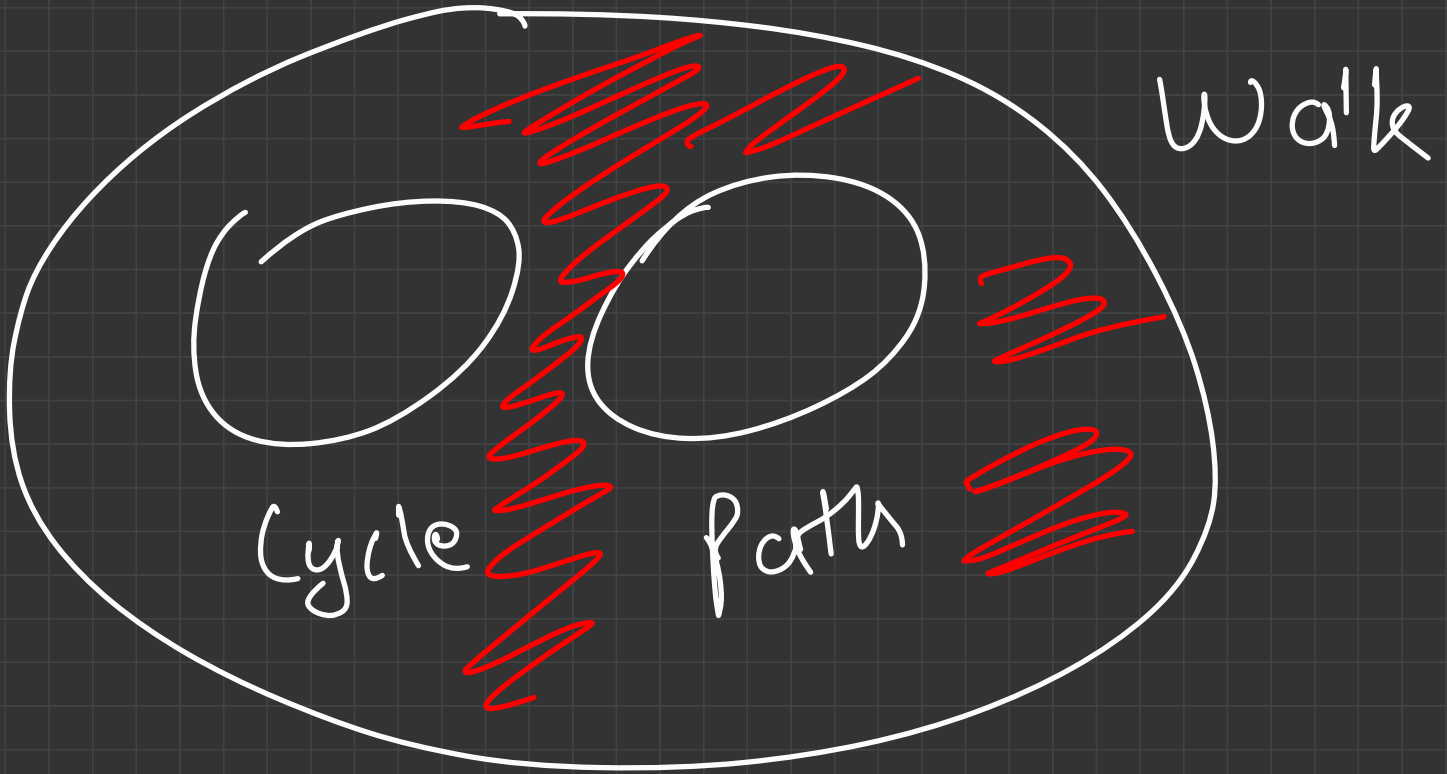
Every path is a walk

Every cycle is a walk


A
B
multi-edge
Split loop



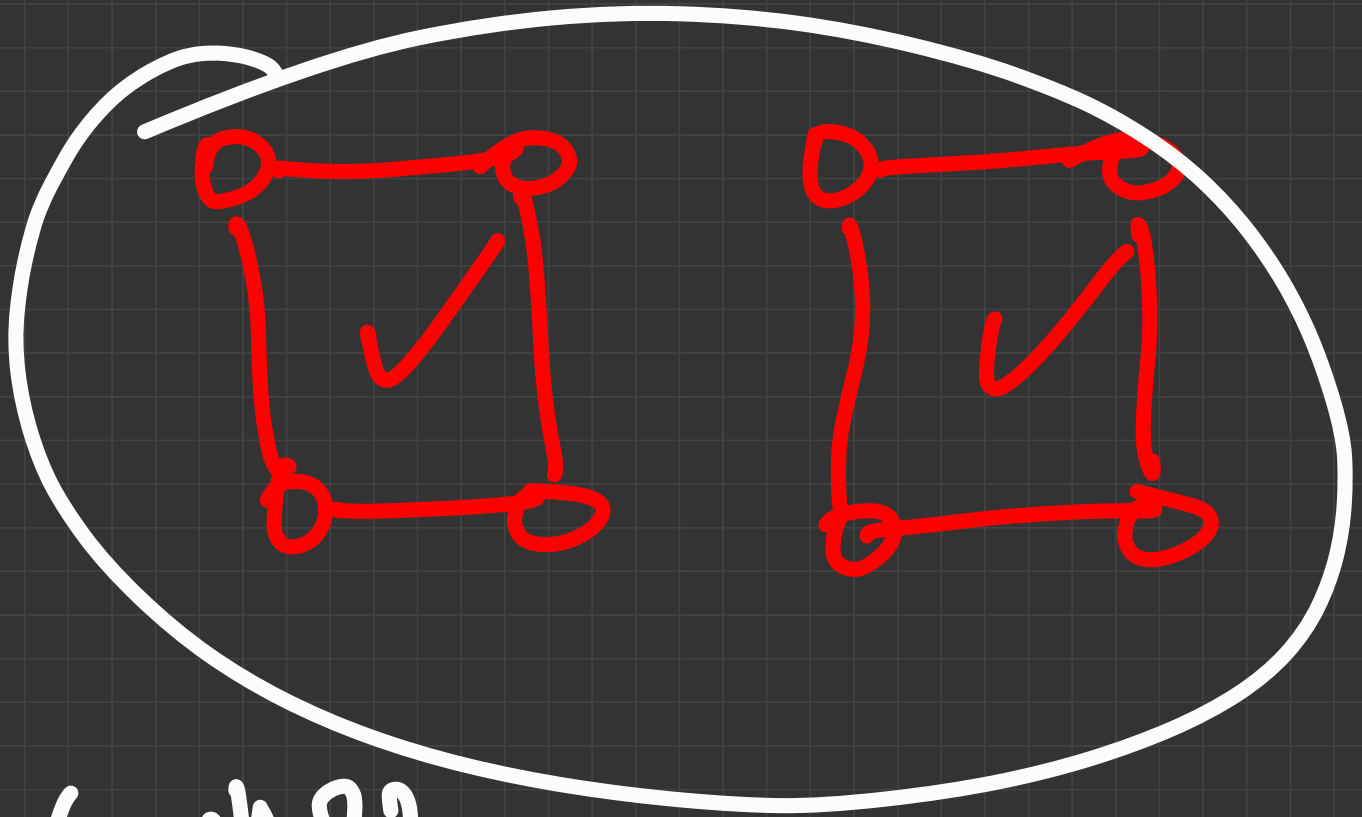
SPH loops &
multi-edge

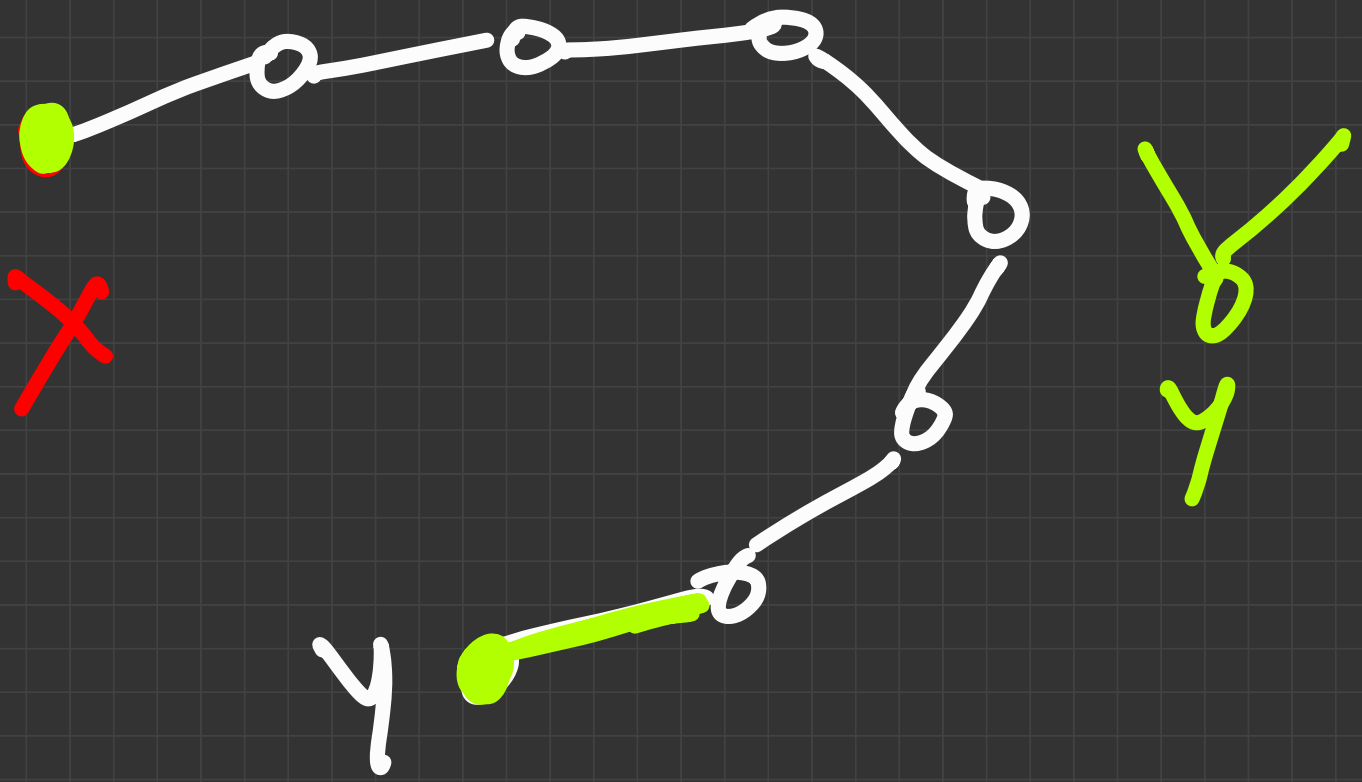

Walk
Cycle Path

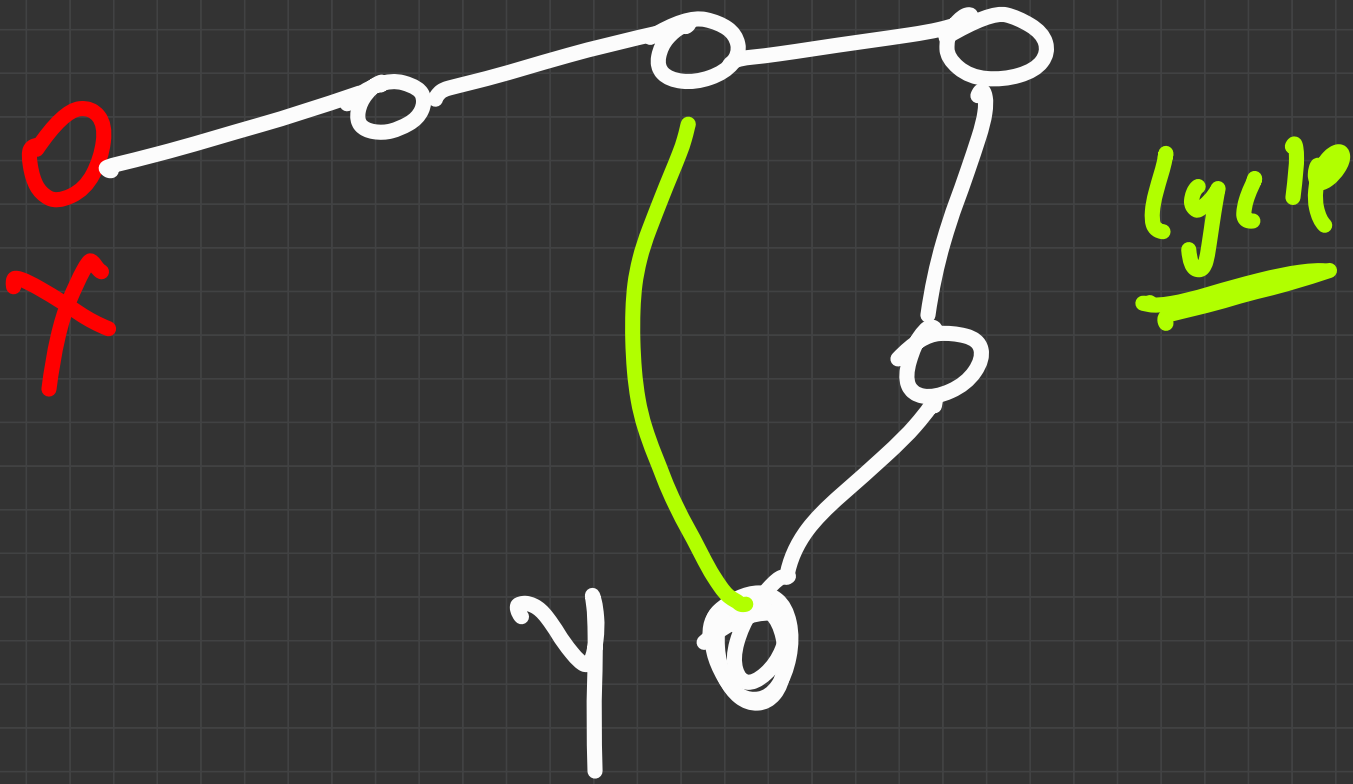
Some Common Results

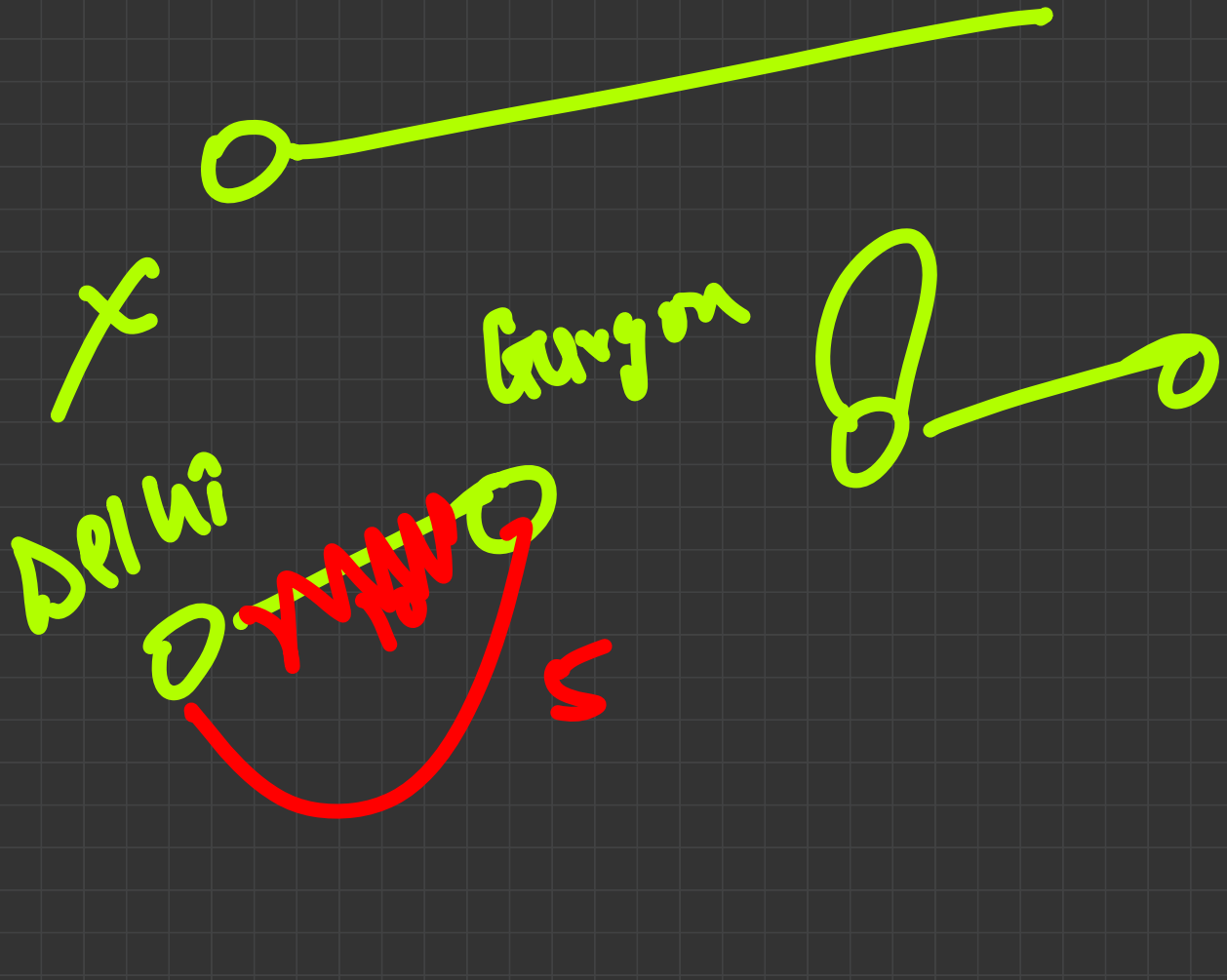
- An undirected graph where each node has at degree at least 2 will contain a cycle

- A directed graph where each node has at least 1 in-degree and at least 1 out-degree will contain a cycle


Graph ??


x
y
y
y


x
y
cycle


x
Gurgaon
Delhi
s

Some Common Results

- The sum of all degrees is even. The number of vertices with odd degree is even.
- Some more as we move ahead...

$$\left[\sum_{i=1}^{n} \text{degree}(i) \right]$$

$$= \text{even number}$$

A

B

+1

+1

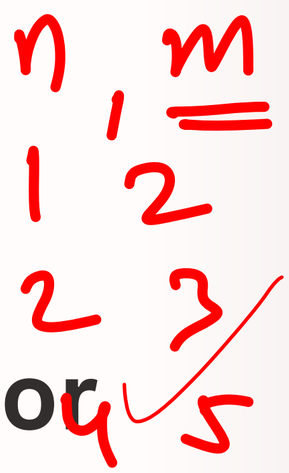
total sum = even

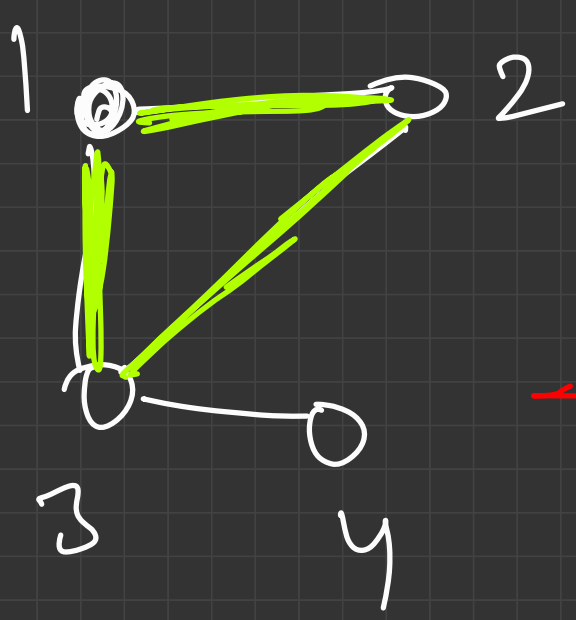
{even + even + even + odd}

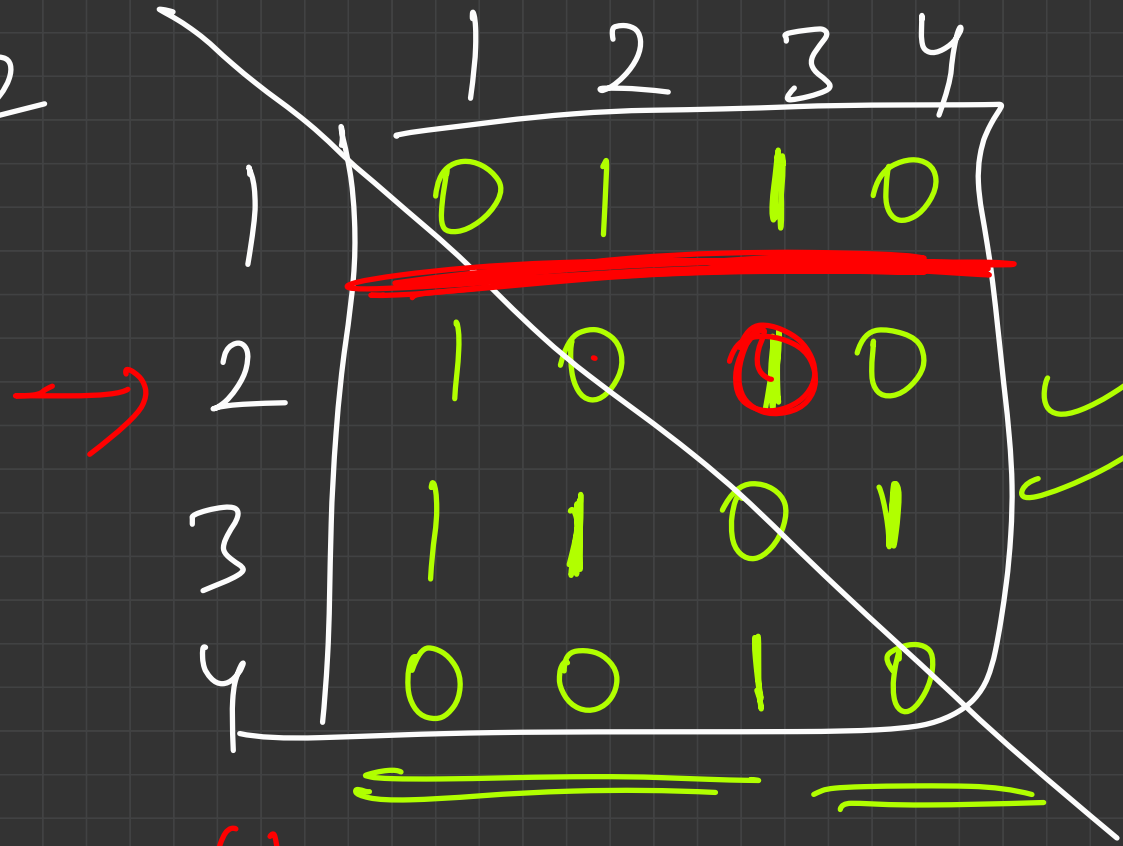
(even + even + odd + odd)

Representation

- Adjacency Matrix
- Adjacency List with Vector
- Adjacency List with Set
- Pros and Cons of each
- ~~How is Input given in problems?~~







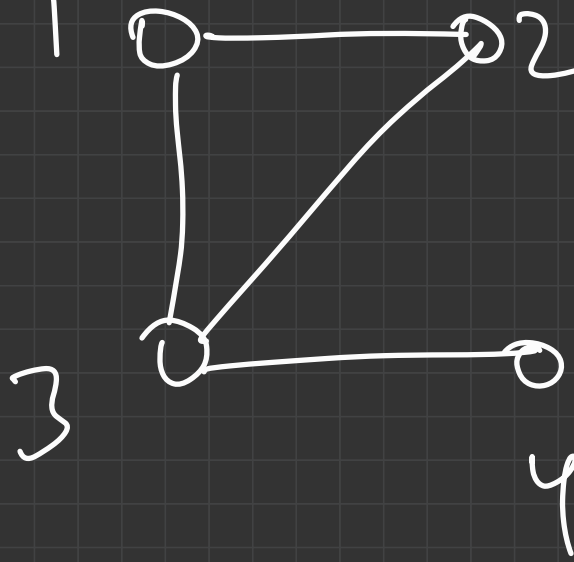
edge (x, y) O(1) edge(2,3) = Yes

Adjacency Matrix

1. Symmetric Matrix
2. Precomputation Time → $O(n^2 + e)$ ≈ $O(n^2)$
3. Space Complexity → $O(n^2)$
4. Find an edge → $O(1)$

⑤ Delete an edge → O(1)

Add an edge → O(1)



1 → 2, 3

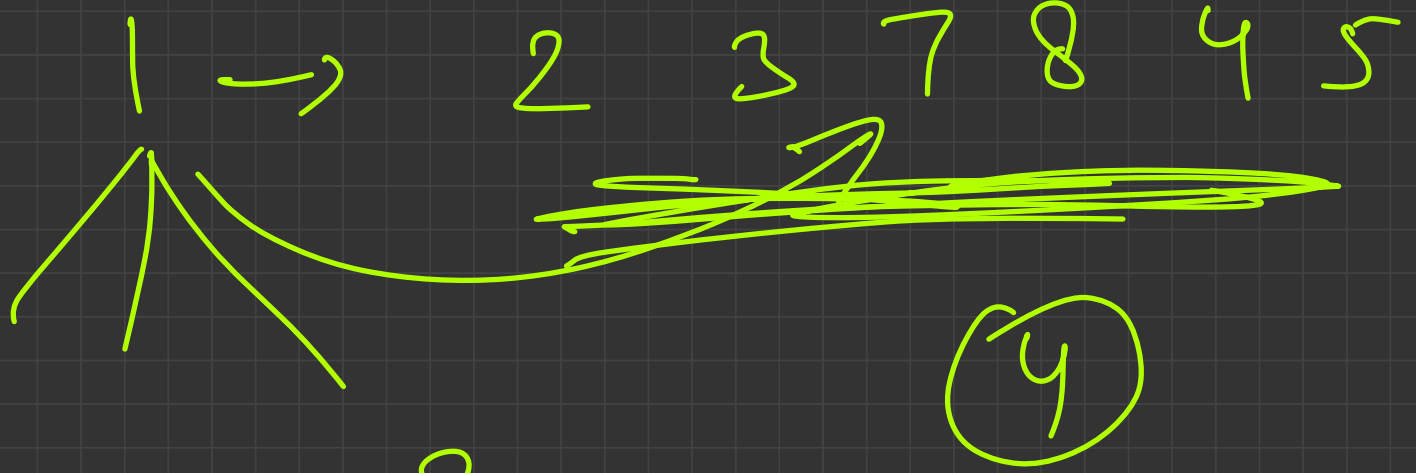
2 → 1, 3

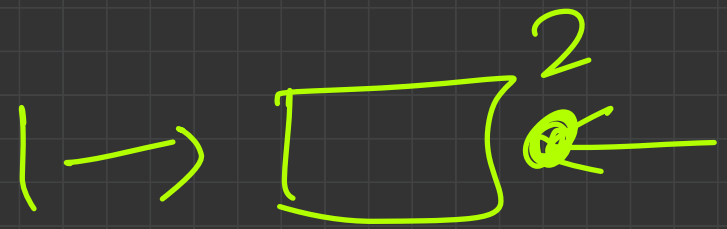
3 → 1, 2, 4

4 → 3

Adjacency list

1. Construction Time → O(n+e)
2. Space Complexity → O(n+e)
3. Checking an edge → O(n)
4. Add an edge → O(1)
5. Delete an edge → O(n)

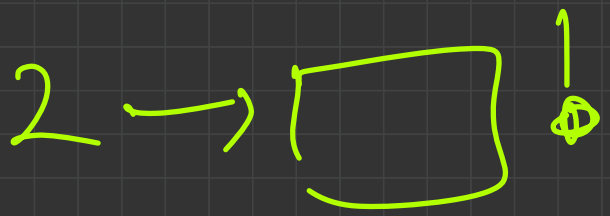


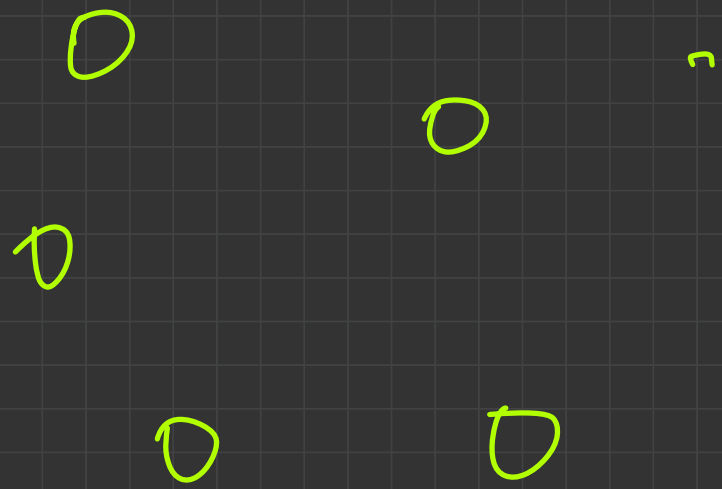


O(n)

O(n)

O(c)





$n \leq 10^5$, $e \leq 10^5$

$n \leq 10^3$, $e \leq 10^5$

Matrix	List
① Construction : $O(n^2)$	$O(n+e)$
② Add edge : $O(1)$	$O(1)$
③ Delete edge : $O(1)$	$O(n)$
④ Checking edge : $O(1)$	$O(n)$
⑤ Space : $O(n^2)$	$O(n+e)$

$O(\log n)$

vector < set < int >> edges (n);

Construction : $O(n) + O(e \log n)$

Add, delete, check → $O(\log n)$

Space complexity : $O(v + e)$

C++

```
vector < vector <int >> edges(n) ;
```

Adj list

Pajowh

```
ArrayList <int> edges[n];
```

```
List < List <int>> (n);
```

vector < set < int >> edges(n);

add edge(1, 3)

O(log n)

edges[1].insert(3)

edges[3].insert(1)

edges[1].push_back(3);

edges[3].push_back(1);

```
vector< set<int> > edges(n);
```

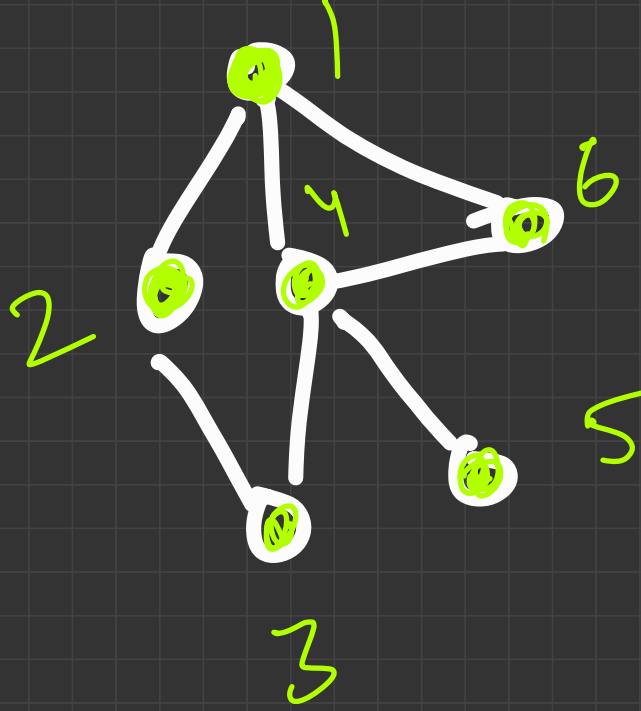
add edge → edges[x].insert(y); O(log n)

remove edge → edges[x].erase(y);

check edge → edges(x).find(y) != edges[x].end()

Traversals

- DFS
- BFS (Single source and Multi source)
- Application of Traversals
  - Connected components (Problem)
  - Path construction
  - Cycle detection
  - Shortest Path (Problem)



parents [-1 1 2 1 -1 1]

1 2 3 4 5 6

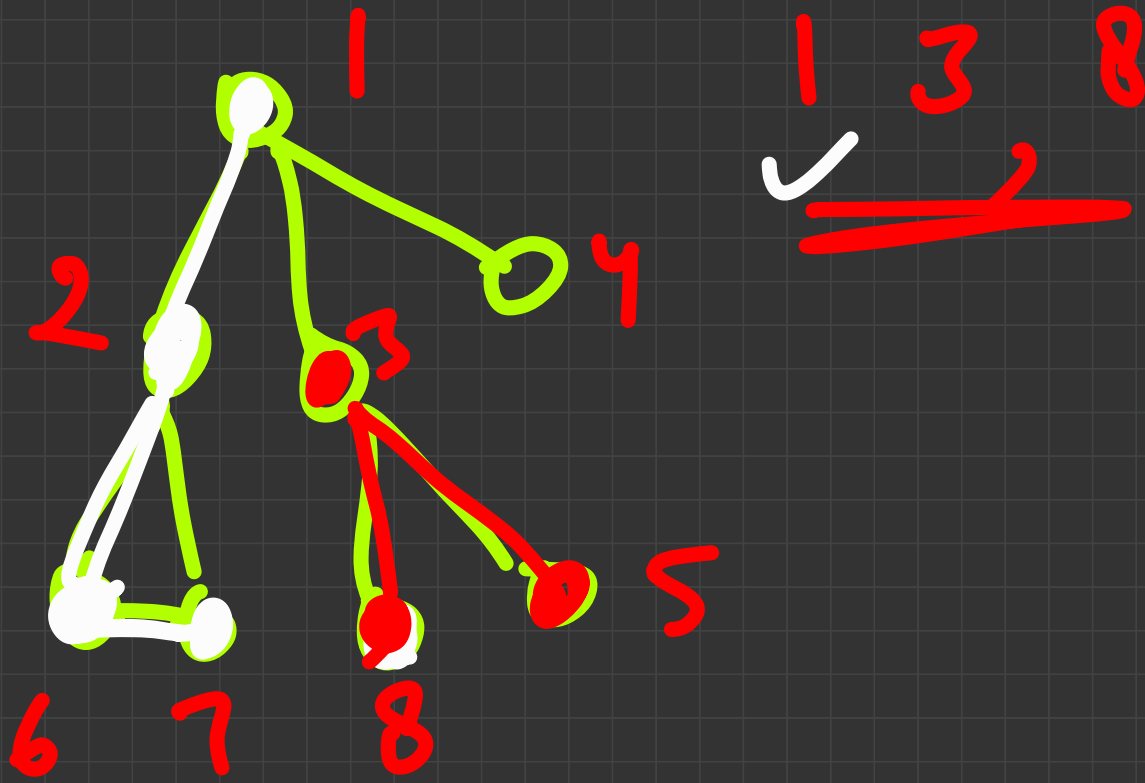
3 → 2 → 1

[1, 2, 4, 6, 3]

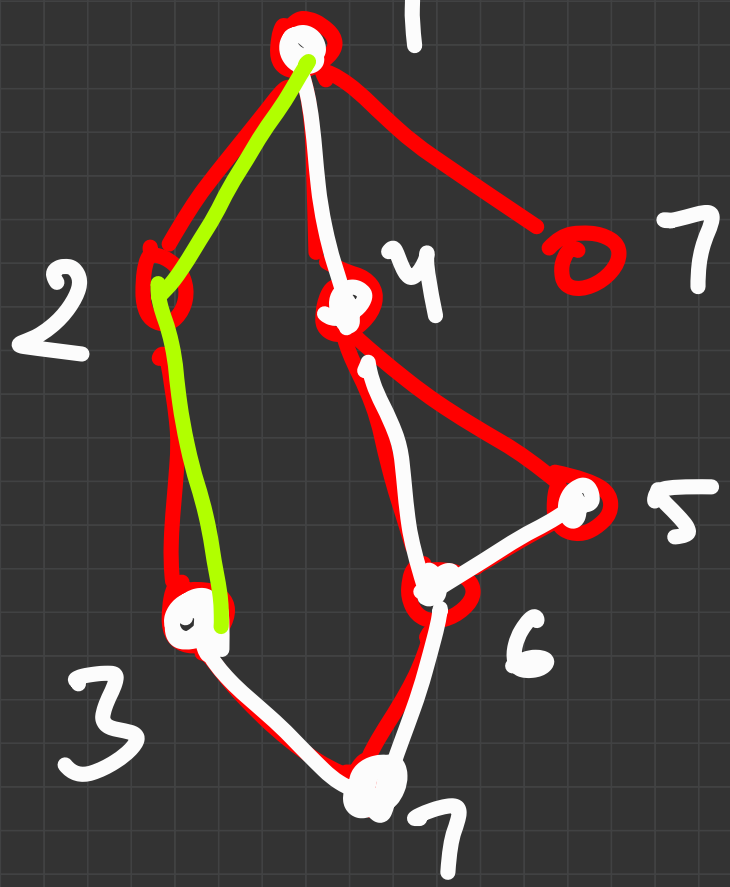
DFS

$A \rightarrow B$

BFS

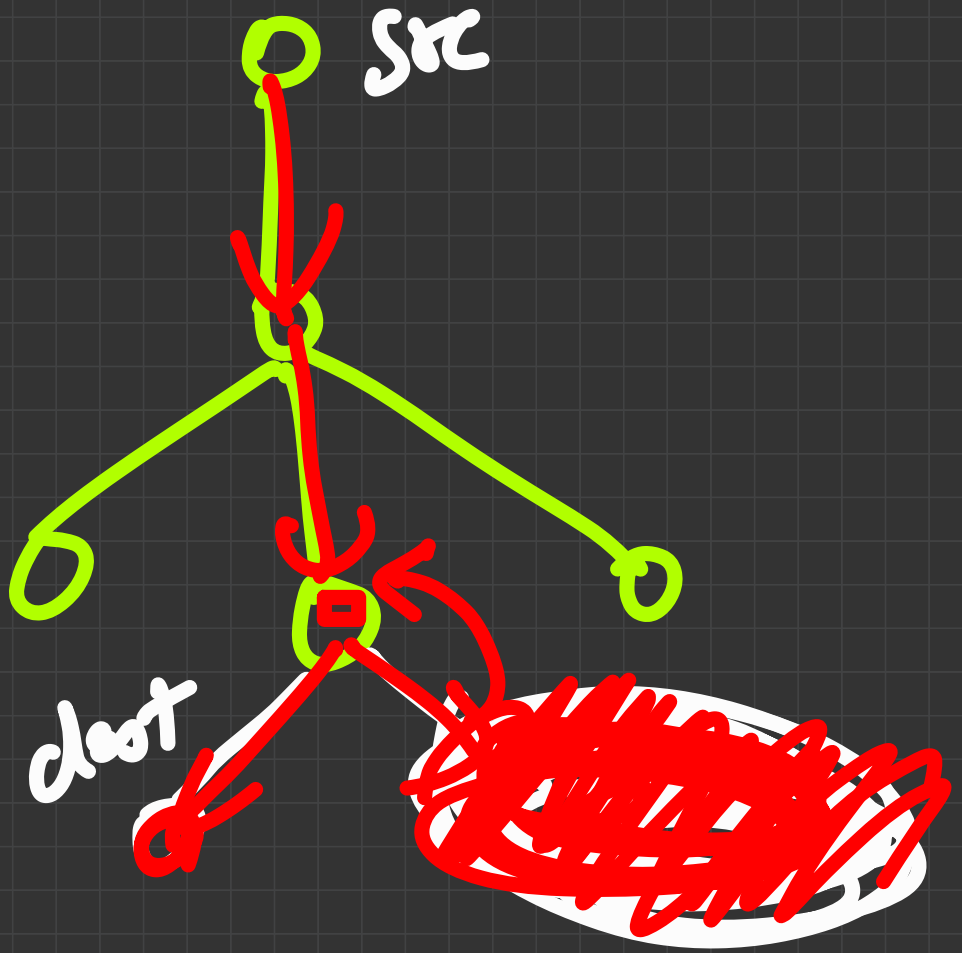


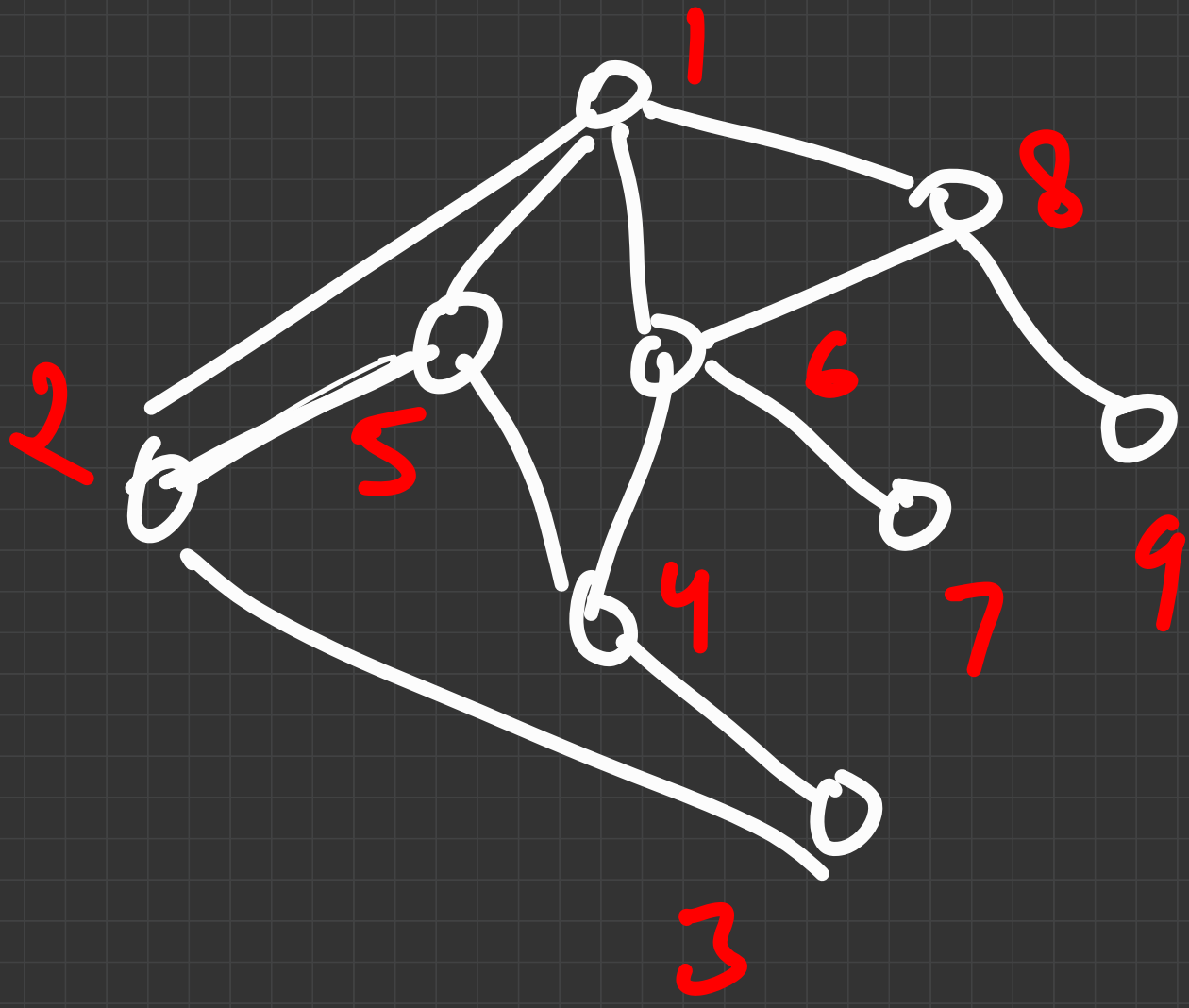
1 3 8

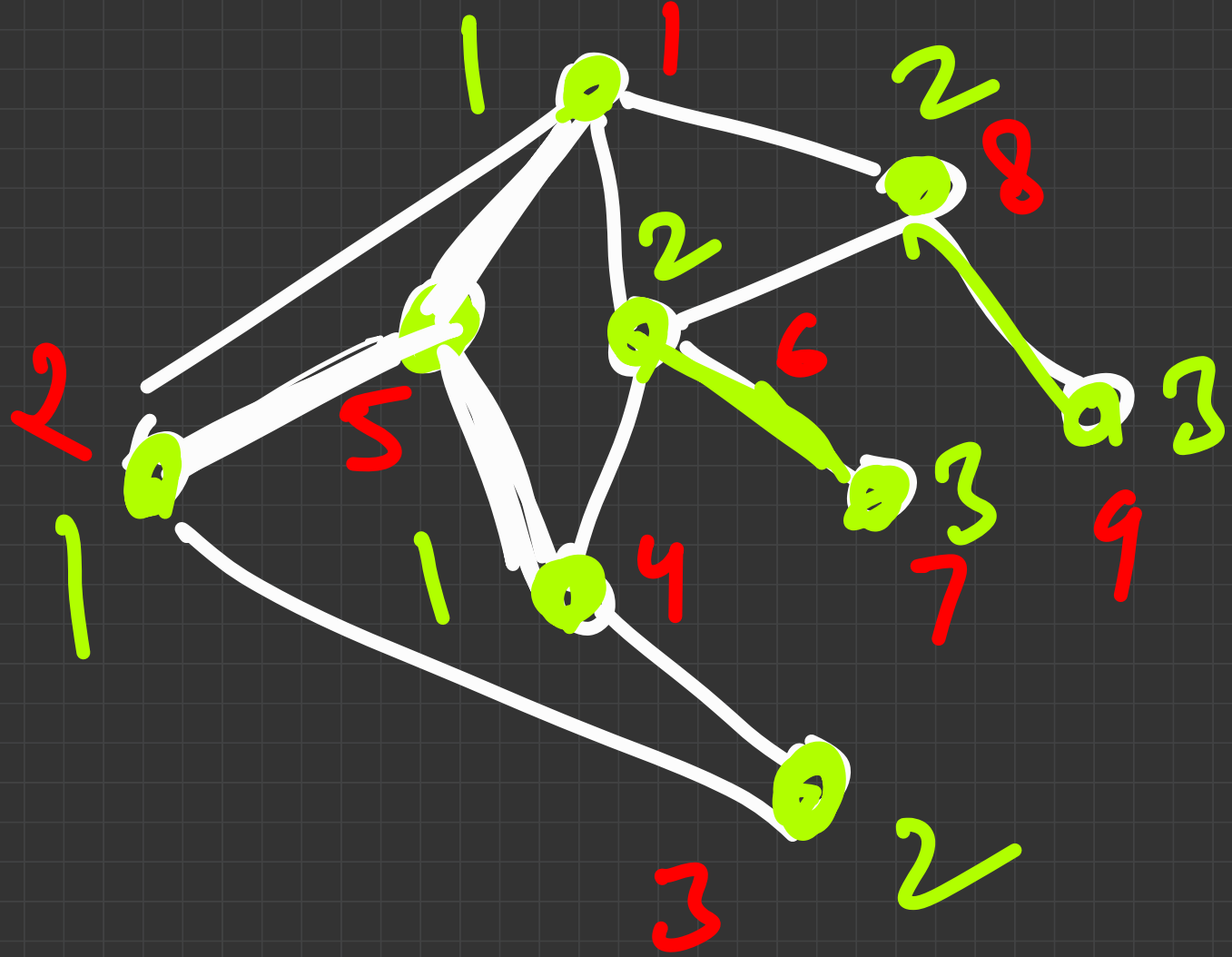


1, 4, 6, 7, 3

1, 2, 3

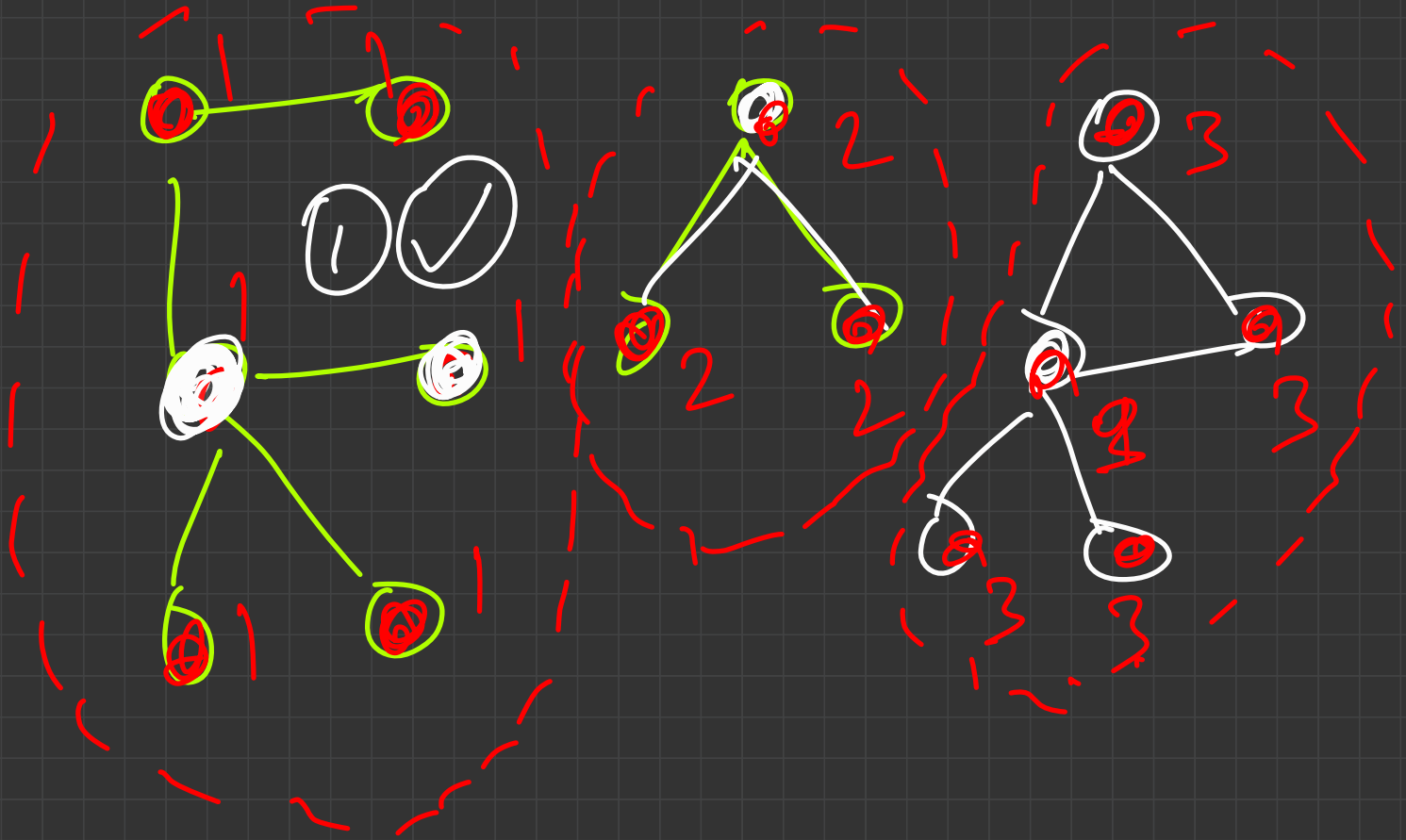

Src
dest

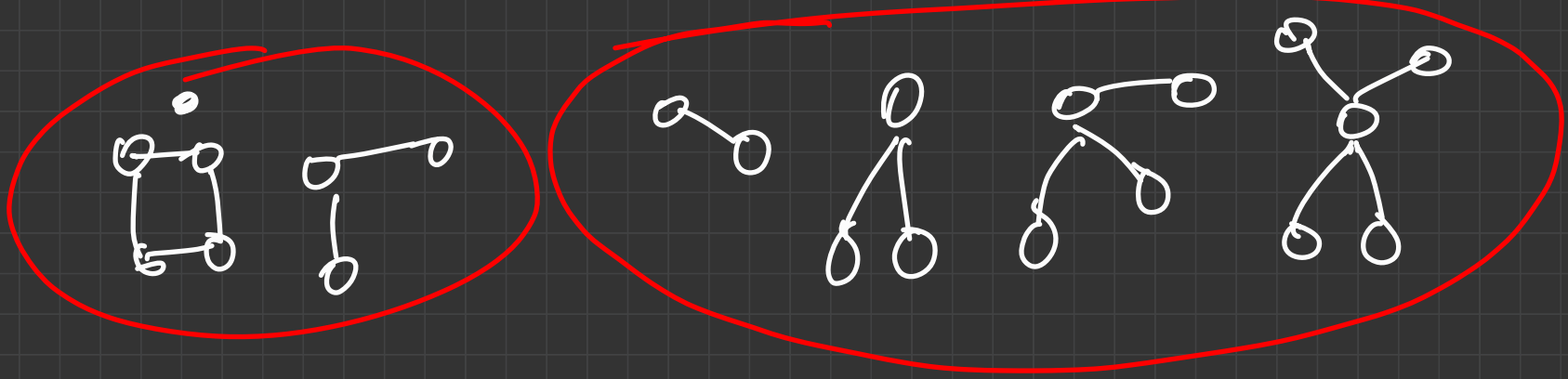

1
8
2
5
6
4
7
9
3


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

```
void dfs(int cur, vector<vector<int>>
        edges, int parent)
{
    for (int neighbour : edges[cur])
        if (neighbour != parent)
            dfs(neighbour, edges, cur)
}
```

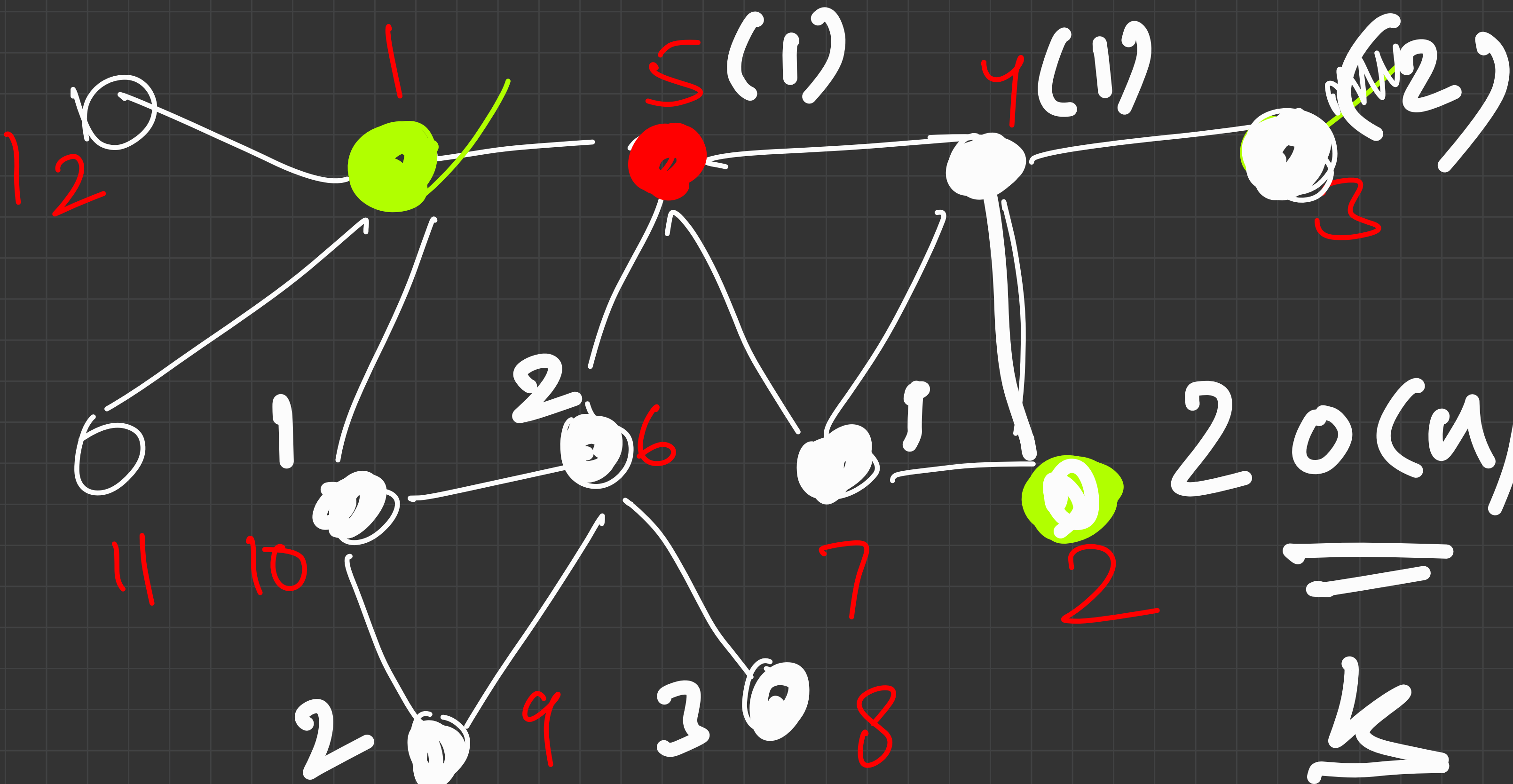
```
void dfs(curr, edges, vector<bool>& vis)
    vis[curr] = true
    for (int neighbour : edges[curr])
        if (!vis[neighbour])
            dfs(neighbour, edges, vis)
}
```





```
int counter = 1
for (int i=0; i<n; i++)
    if (!visited[i]){
        dfs(i, edges, vis, counter)
        counter++; }
```

Multi-source BFS



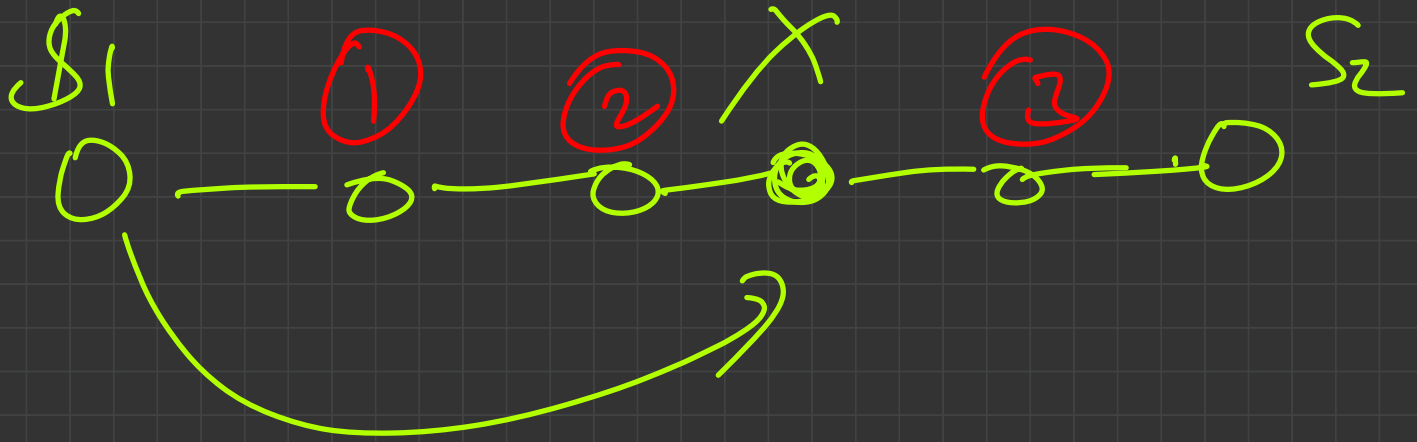
1 2 3 12 1

```
vector<int> dist(n, int), vis(n,0)
queue<int> q;
for( node : special nodes){
        q.push(node)
        dist(node)=0, vis(node)=1
while(!q.empty())
{
```

```
int cur = q.front();
q.pop();
for(x : neighbors of(cur)){
    if(!vis(x))
    {
        dis(x) = dis(cur)+1
        vis(x) = 1
        q.push(x)
    }
}
```

Normal BFS

push the first node into queue



S_1 S_2 1 , 3