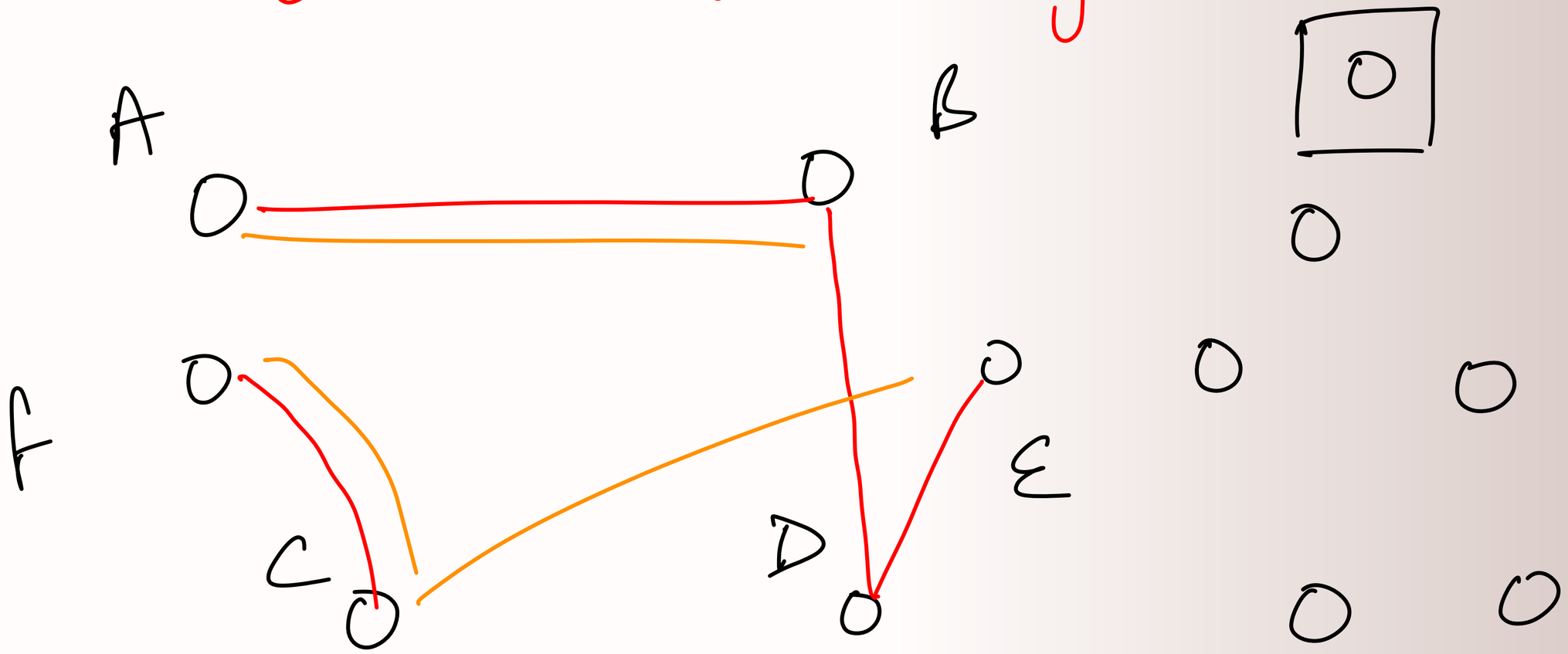
Graphs Class 1

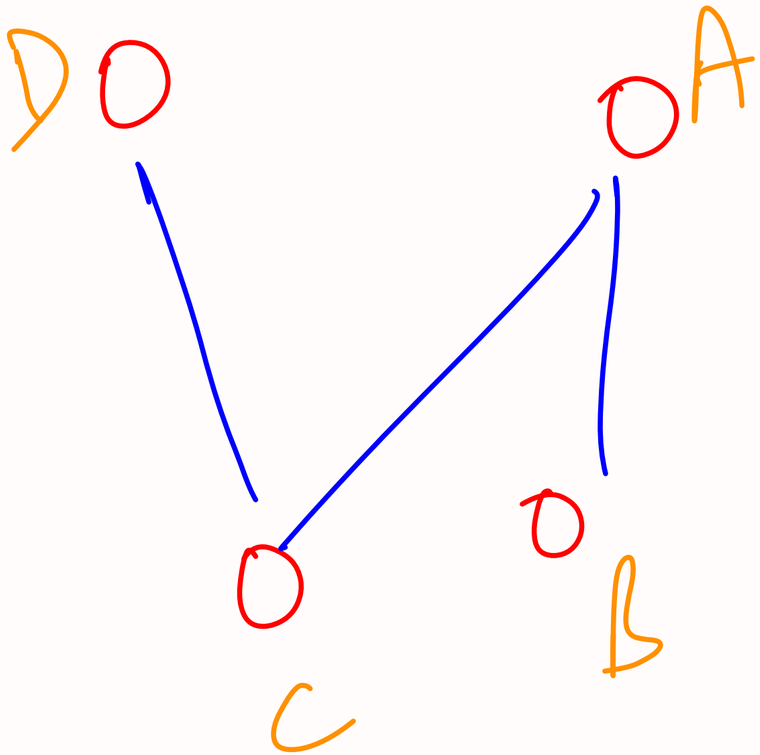
Types of Graphs

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

Nodes connected with edges

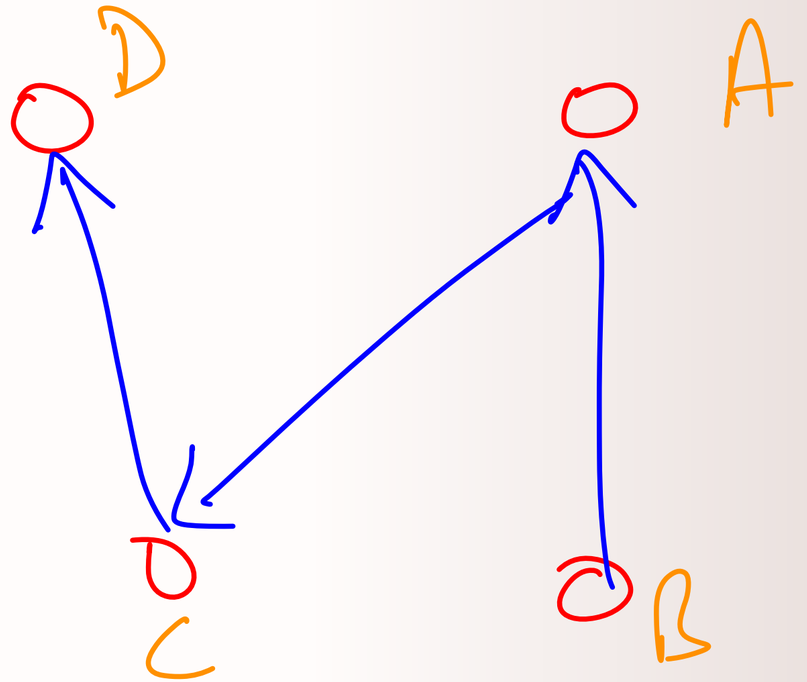


(what is a graph!)



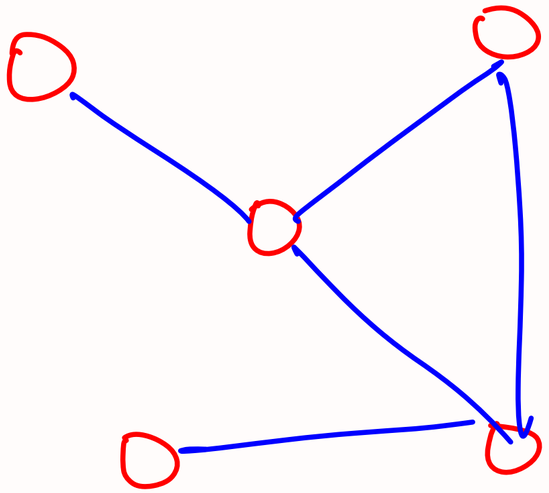
Undirected

(edge b/w two nodes)

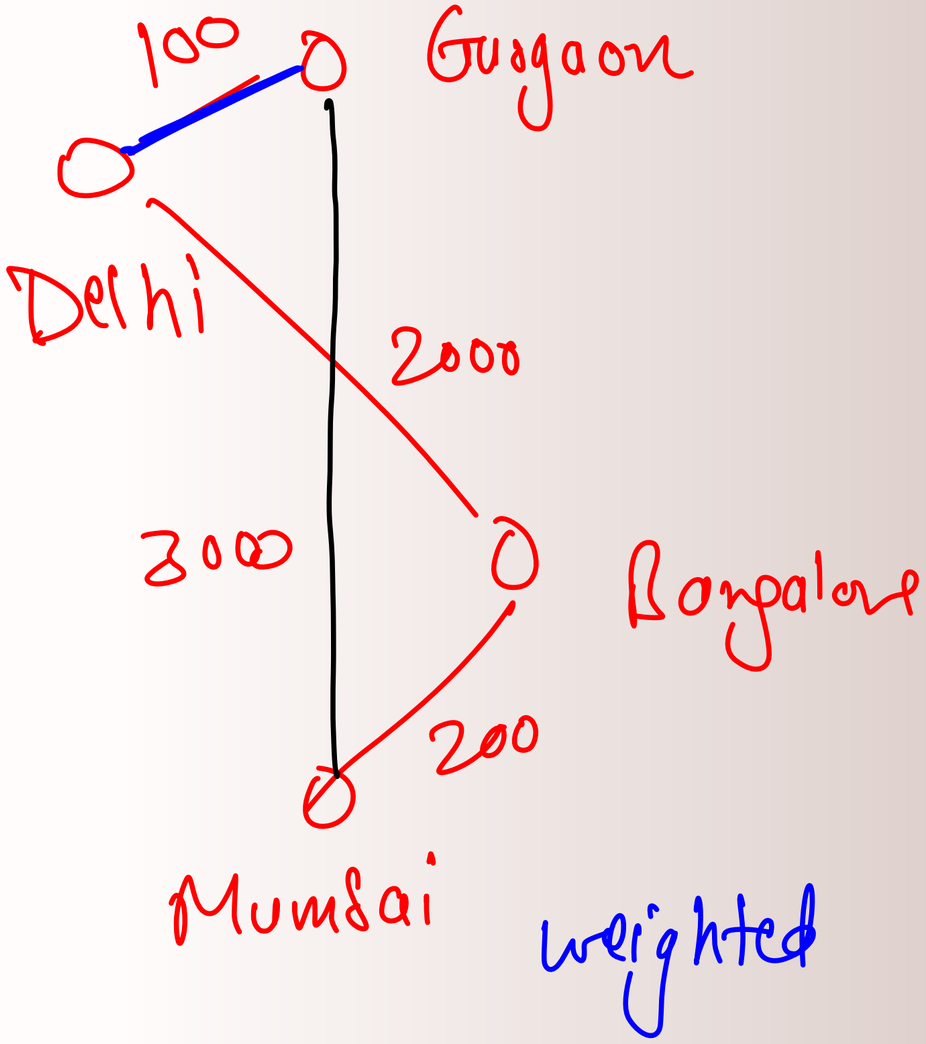


directed

(edge from one node to another)

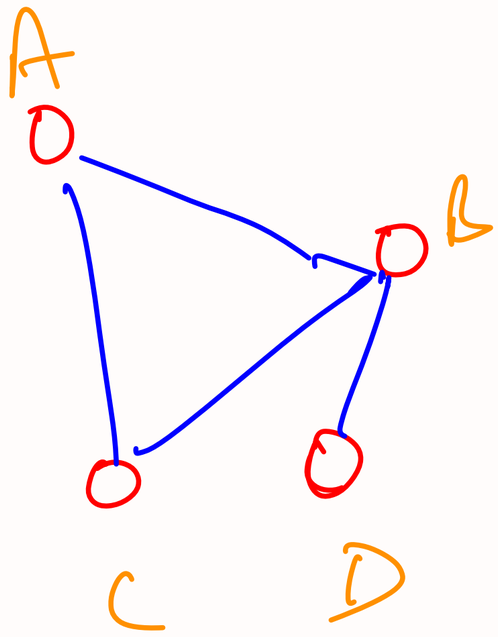


unweighted

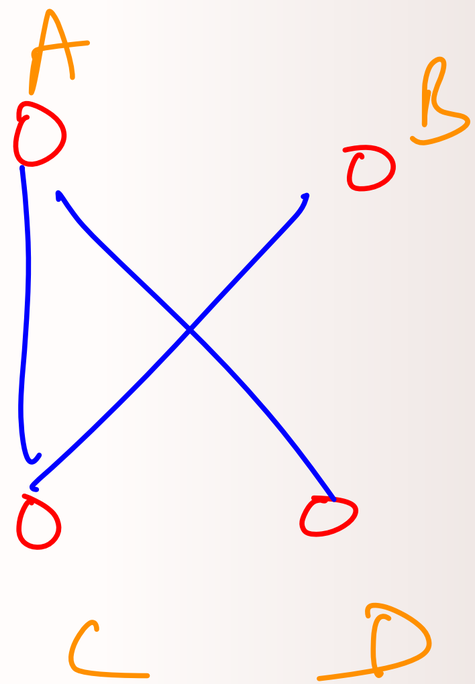


weighted

Cyclic Graph : There are more than 1 paths
b/w two nodes

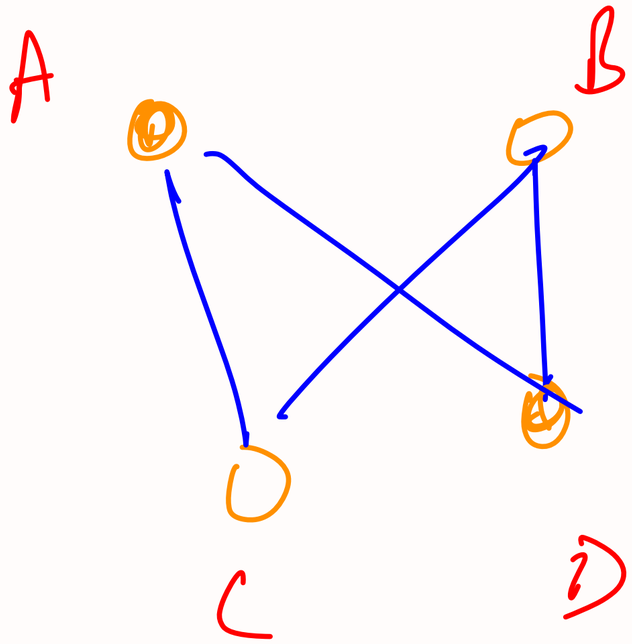


Cyclic

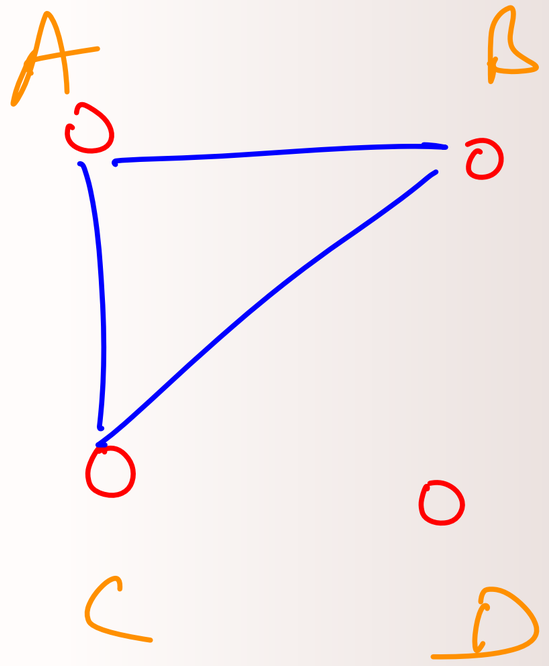


Acyclic

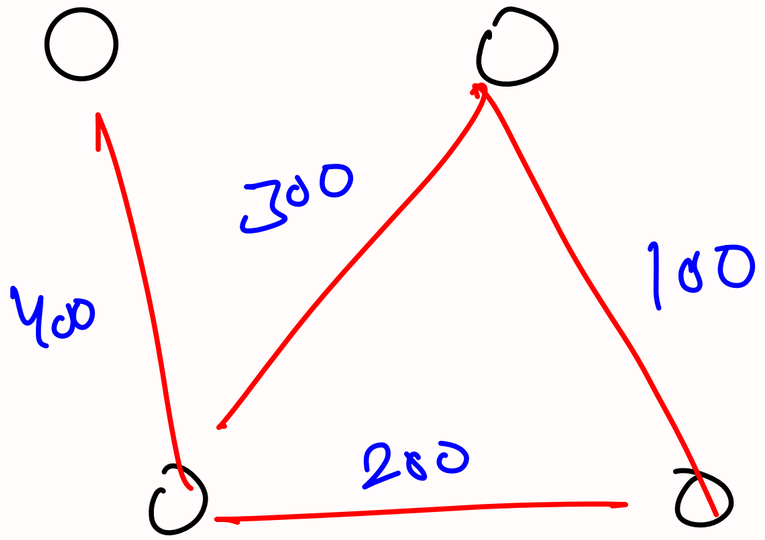
Connected Graph : where you can reach any node from any other node



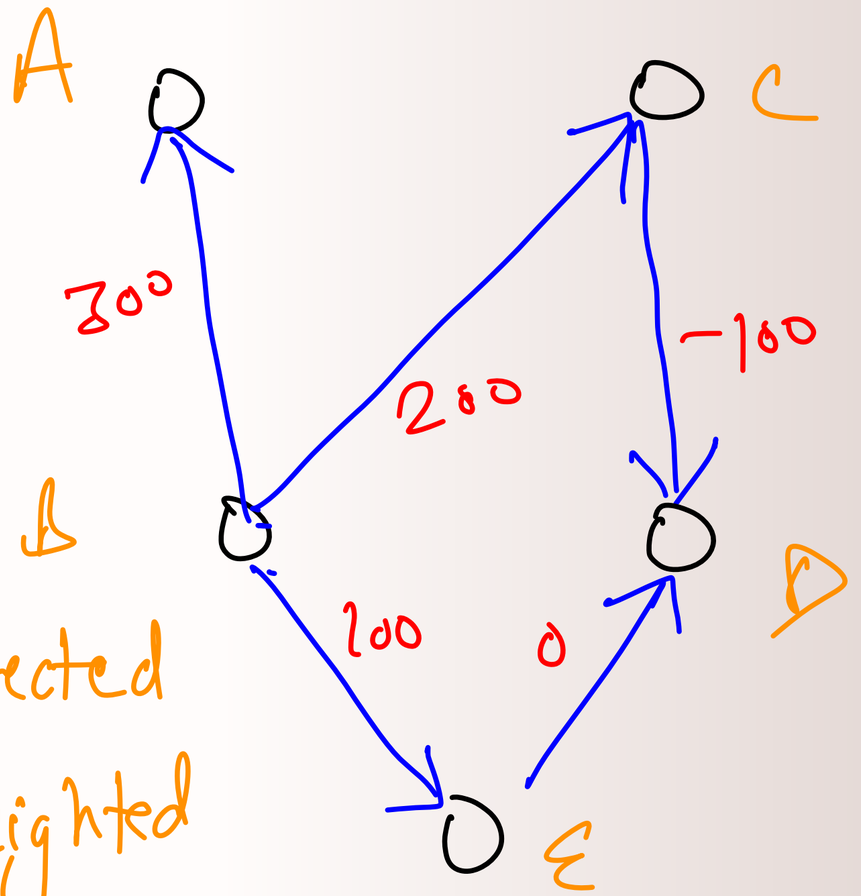
Connected



disconnected



- undirected
- weighted
- cyclic
- connected

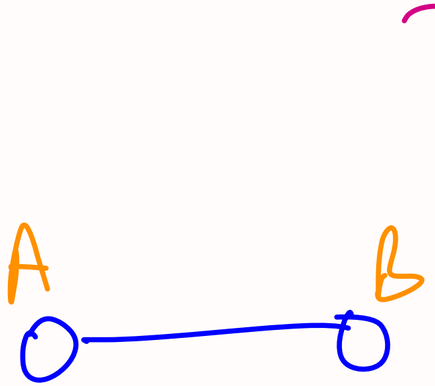


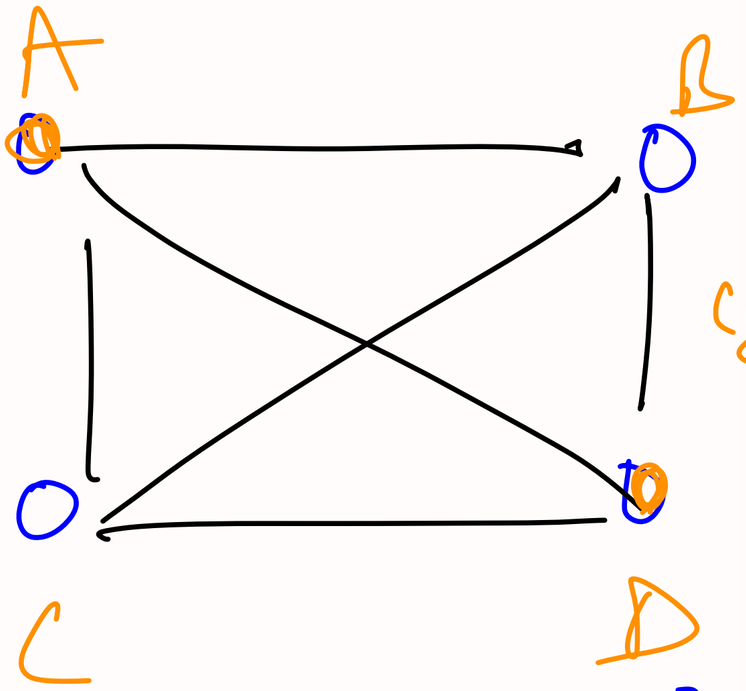
- directed
- weighted

- cyclic vs acyclic
- connected vs disconnected

strongly connected

Complete Graphs : every pair of nodes has a direct edge





cyclic

Q) is a complete graph cyclic or acyclic

Q) how many edges in a complete graph of n nodes

$\frac{n \cdot (n-1)}{2}$ → $^{n}C_{2}$

→ cyclic if no. of nodes > 2

Common Terms

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

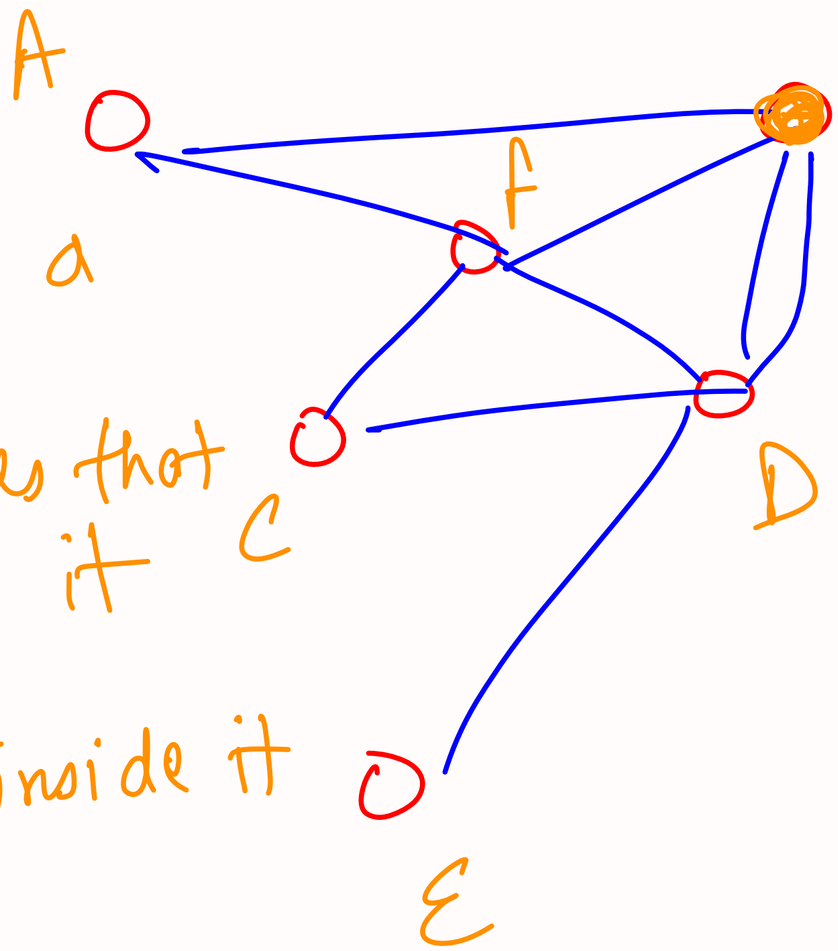
nodes

connections b/w nodes

for directed graphs

indegree

outdegree



neighbour / degree

$N(B) = \{A, f, D\}$

degree of a node = no. of edges that go out of it or that come inside it

degree of a node = no. of neighbour

∝∝ doesn't work for a graph

$N(x)$ = set of nodes that x is directly connected with

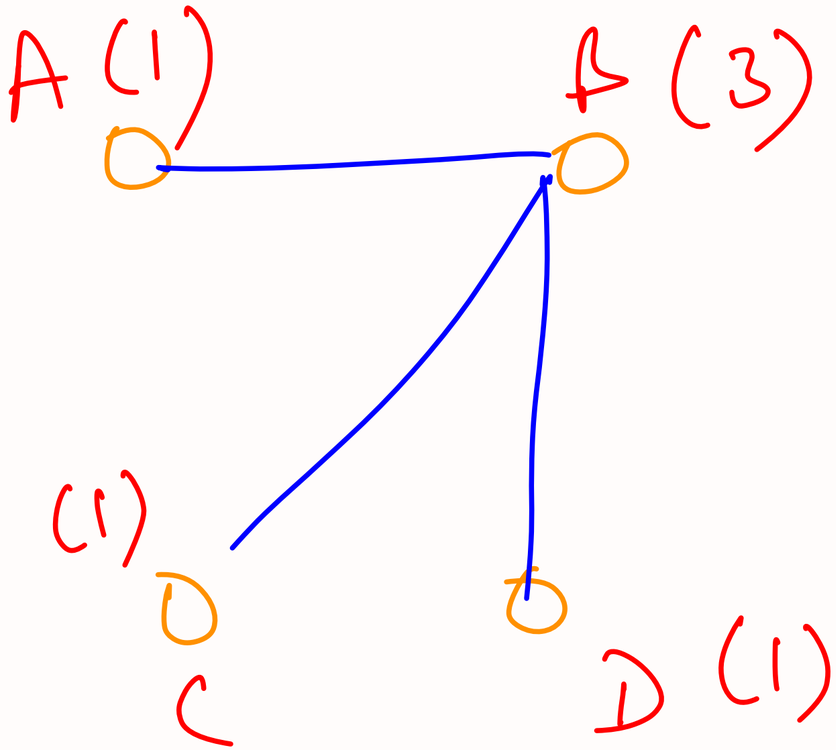
Degree

① $deg(x)$ = no. of neighbours of x

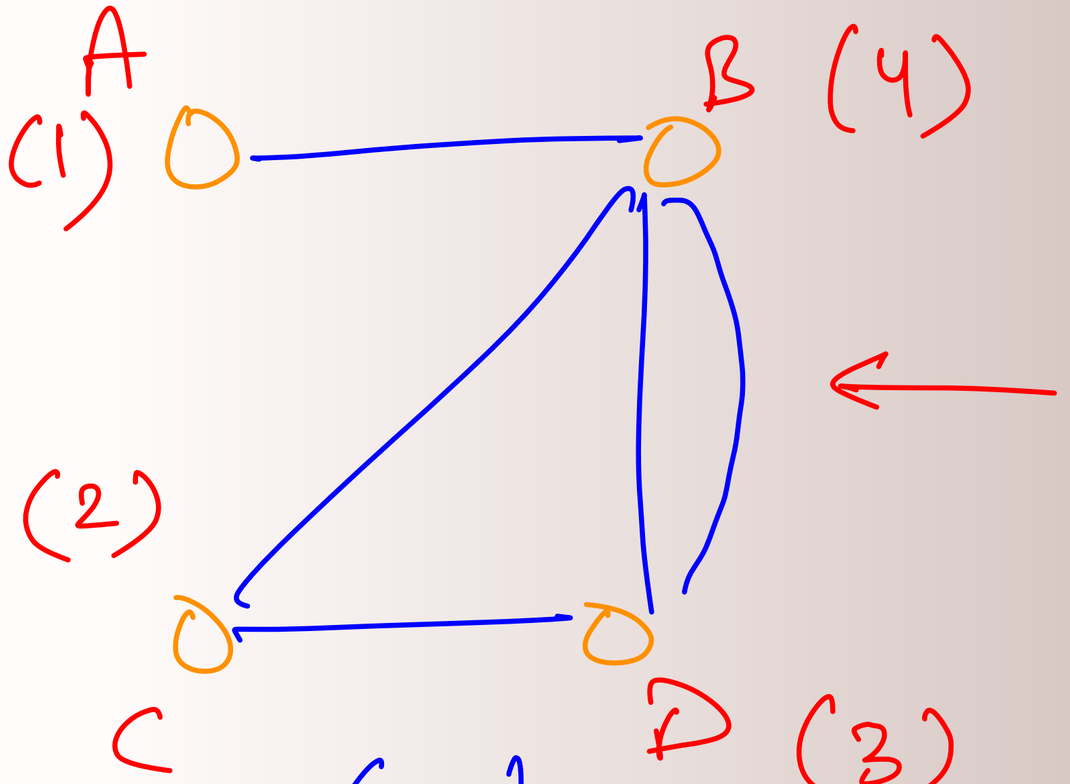
only works for trees

general definition

② $deg(x)$ = no. of edges that go out of a node or come into a node

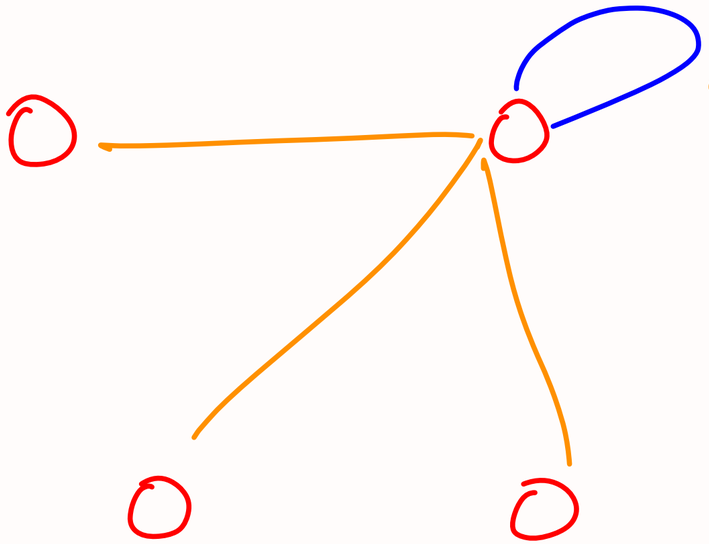


Tree + Graph



Graph

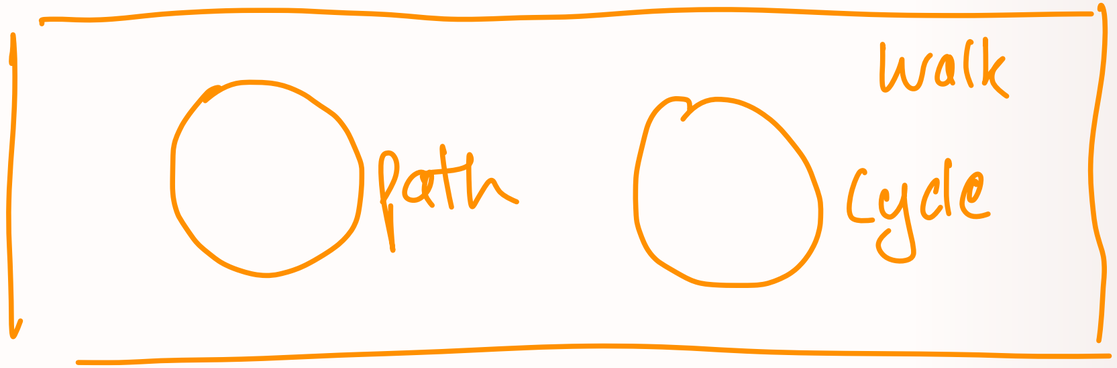
for a tree the no. of edges going out of a node = no. of neighbour of that node



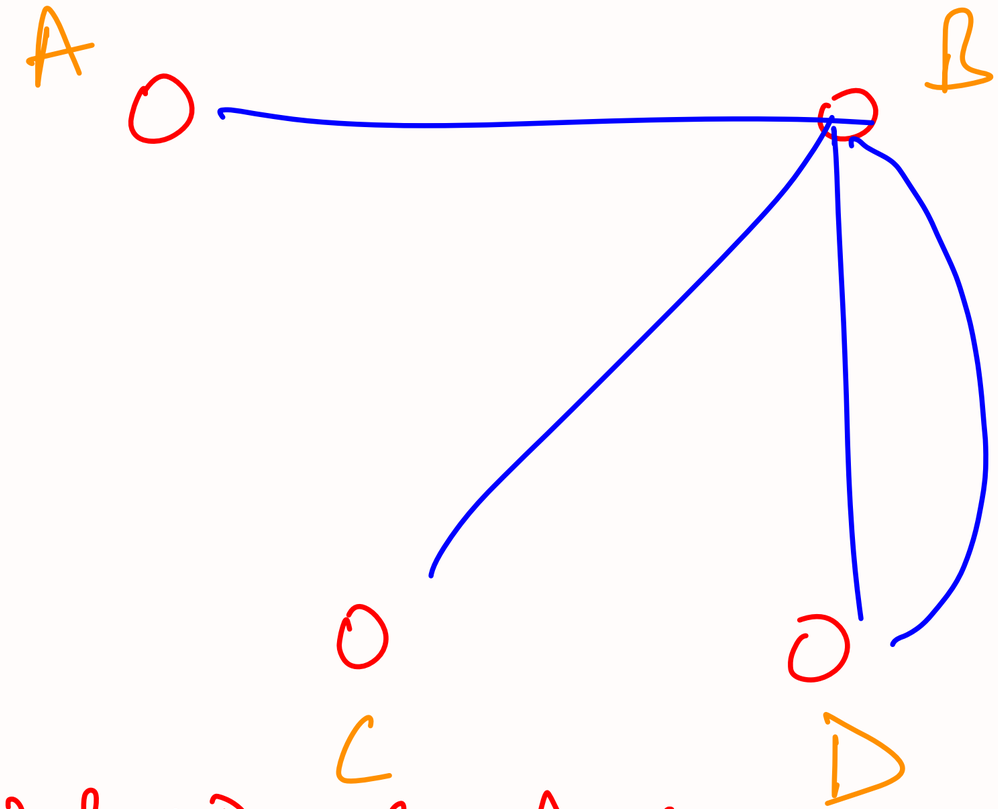
self loop

↳ an edge connecting a node to itself

Path vs Walk vs Cycle



(movements in a graph)



$A \to B \to D \to B \to A$ (walk)

$A \to B \to D \to B$ (walk)

$A \to B \to D$ (path)

$B \to D \to B$ (cycle)

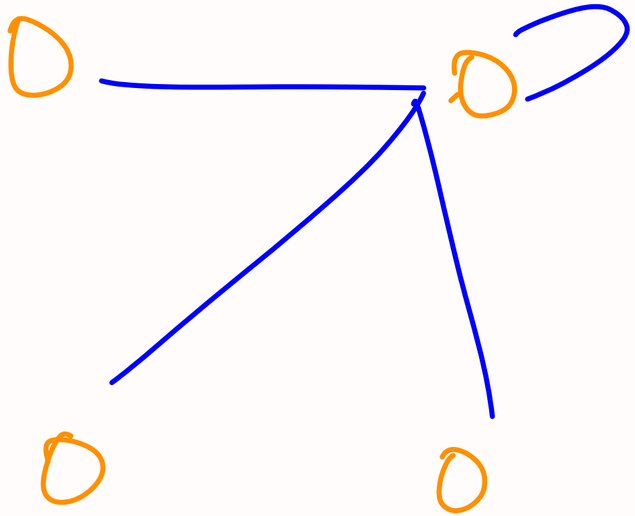
① any movement done in a graph is called a walk

② a path only encounters unique nodes and edges while moving

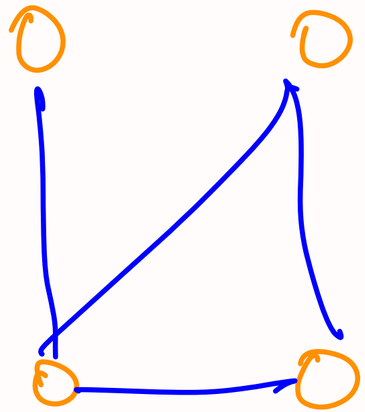
③ a cycle starts from one node and comes back to the same node using unique nodes and edges. only start node = end node

Simple Graph → graph which does not contain any self loops and does not contain multiple edges blw any pair of nodes

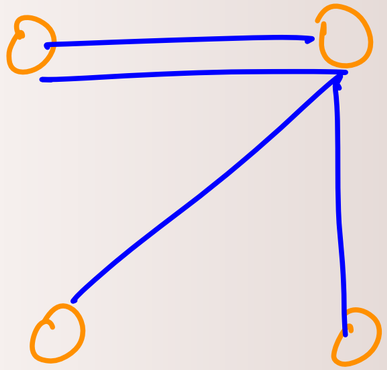
(dy(x) = no. of neighbours of x in a simple graph)



✗✗



✓✓

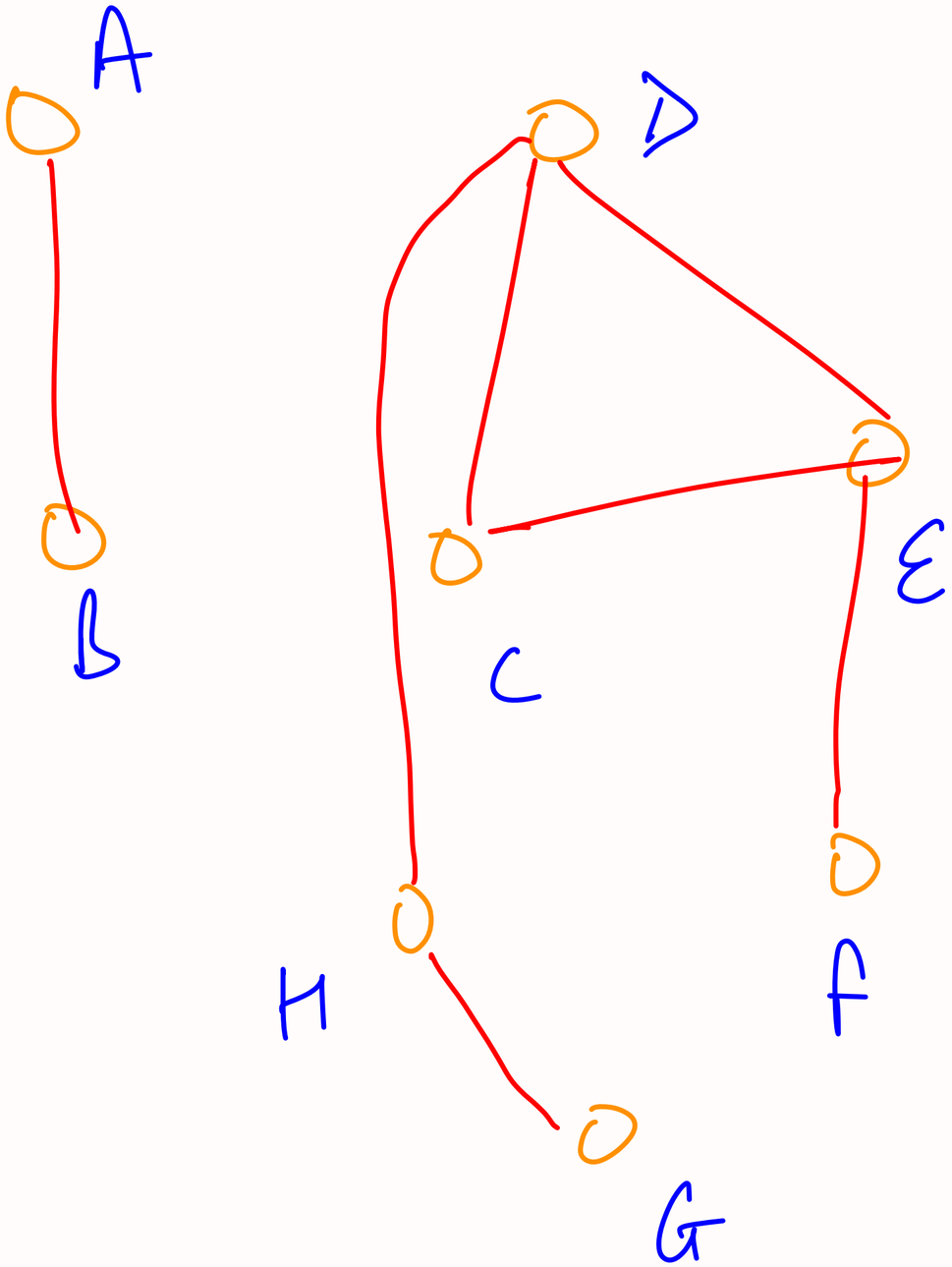


✗✗

Bridge = any edge that when removed from the graph increases the no. of connected components

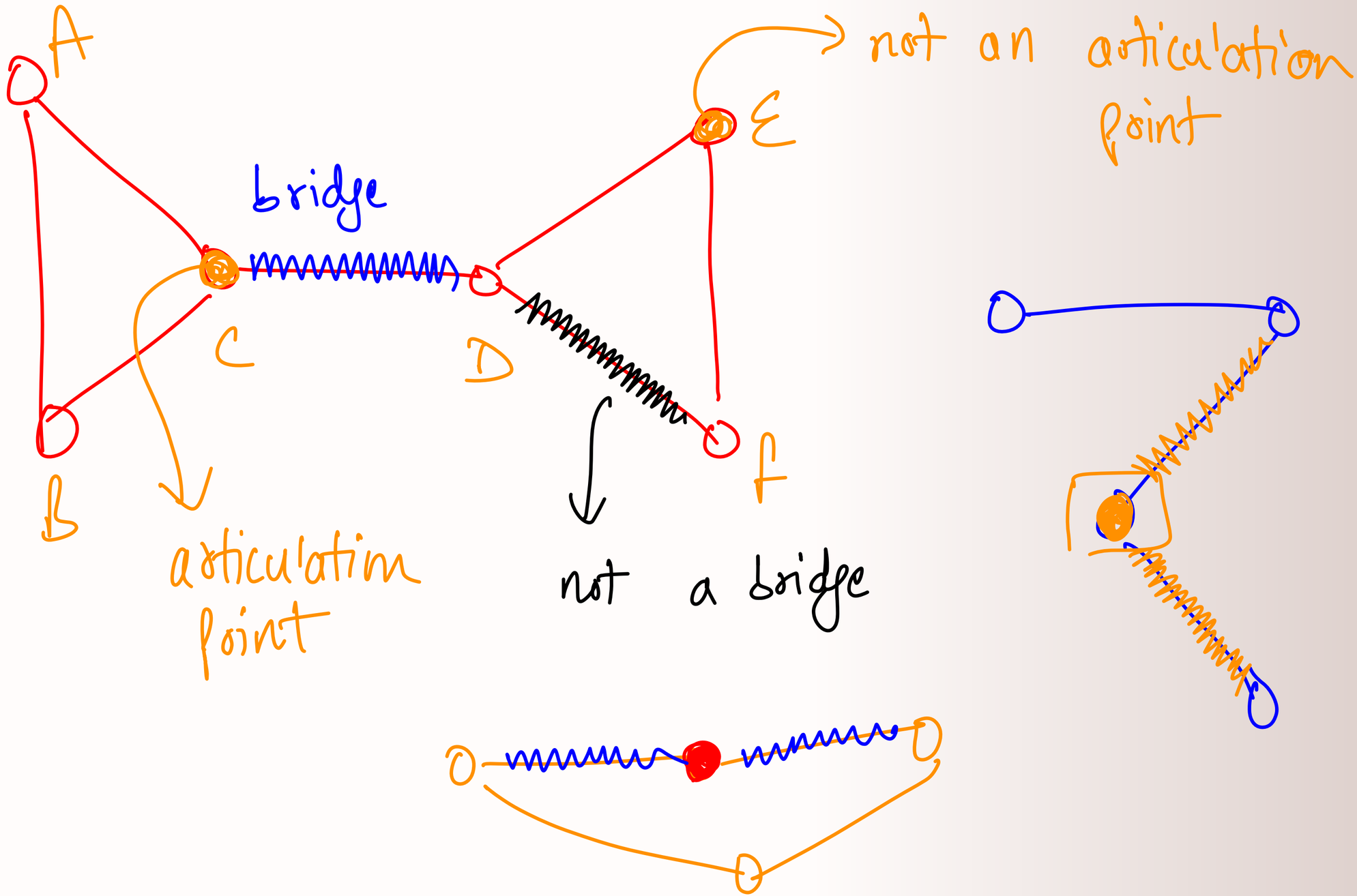
Articulation point = any node that when removed from the graph increases the no. of connected components

connected component !



a connected comp within a graph is a set of nodes that are connected.

a disconnect graph has more than 1 connected component


A
B
C
D
E
F
bridge
not an articulation point
articulation point
not a bridge

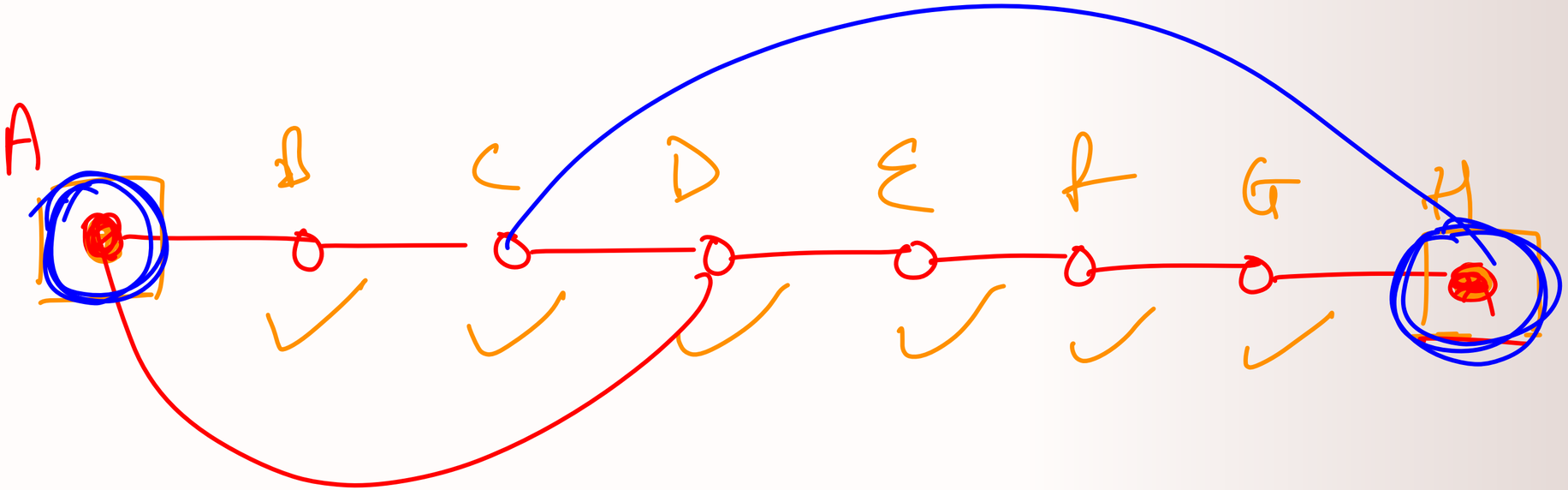
Some Common Results

- An undirected simple 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

when we study directed graphs

every node has degree at least 2



(undirected + simple graph)

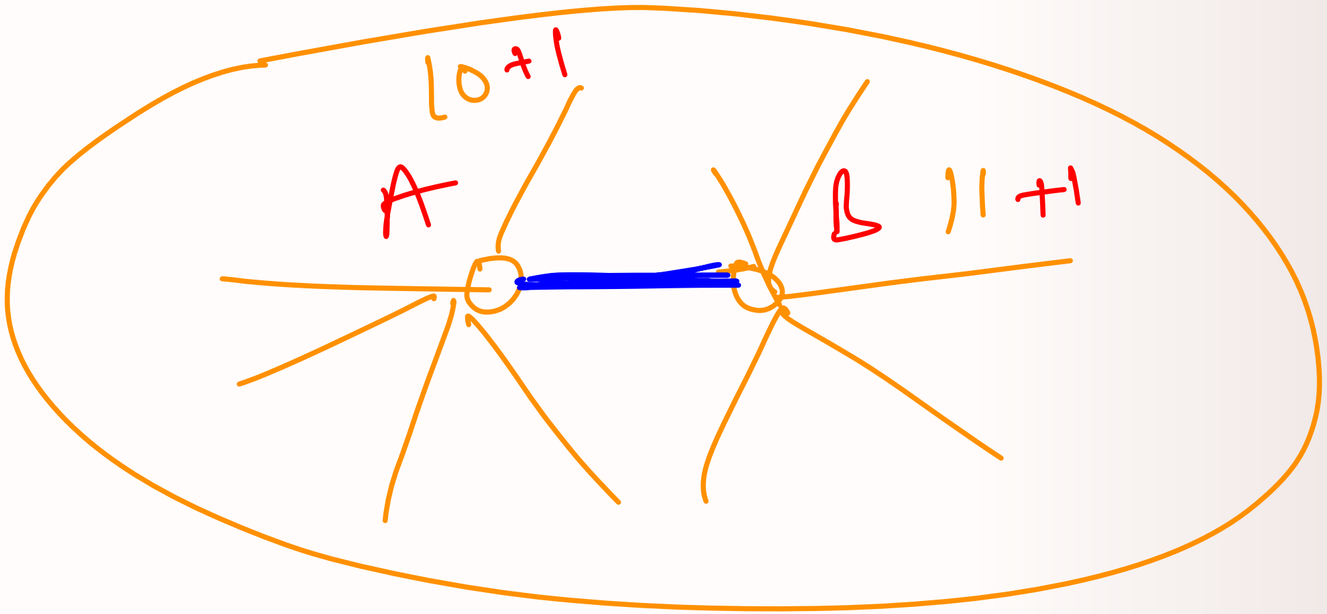
Some Common Results

- The sum of all degrees is even. The number of vertices with odd degree is even.

- Some more as we move ahead...

$$\sum_{i=1}^{n} deg(i) = \text{even}$$

$$\sum_{x=1}^{n} \deg(x) = \text{even}$$



every edge contributes +2 to the total degree sum.

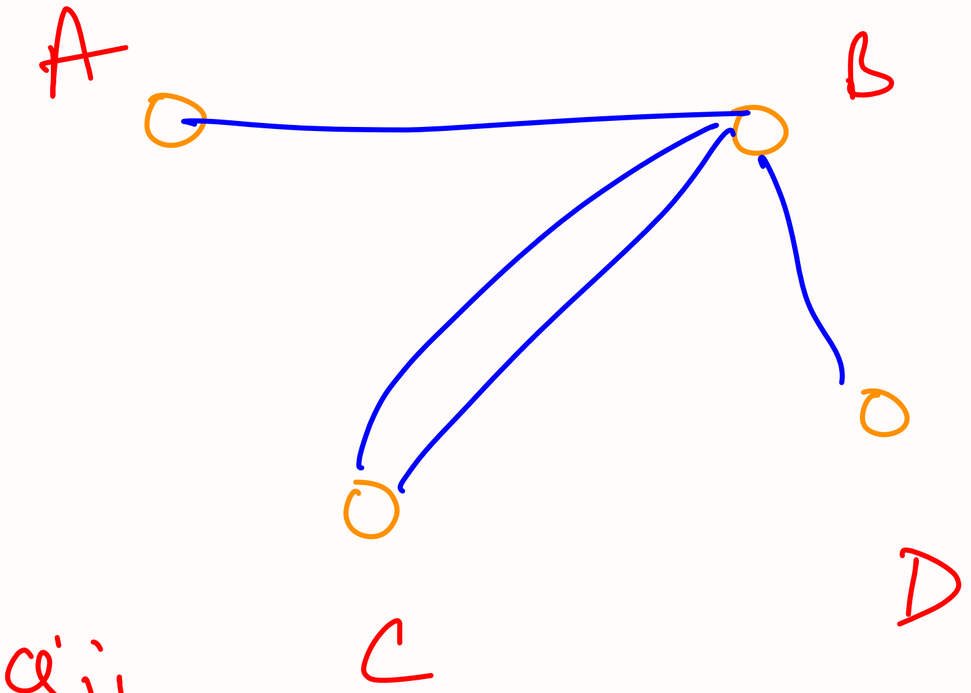
$$DS = d_1 + d_2 + d_3 \cdots d_n$$

DS ↓ even

$d_1 + d_2 + d_3 \cdots d_n$ ⌣ no. of odd terms is also even

Representation

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

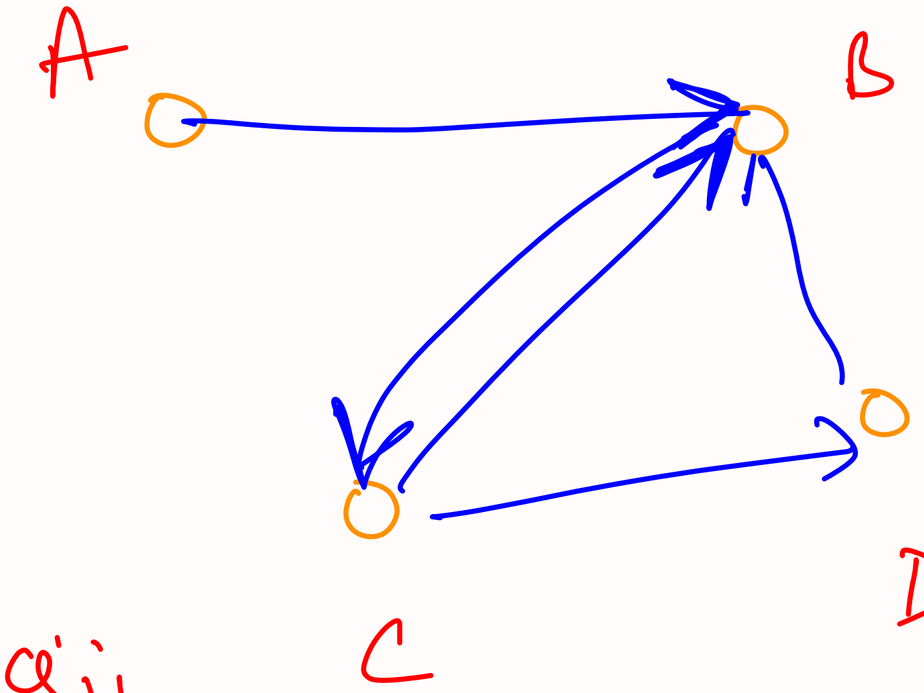


$a_{ij} = a_{ji}$

$\forall i, j$

	A	B	C	D
A	0	1	0	0
B	1	0	2	0
C	0	2	0	1
D	0	1	0	0

Adjacency matrix storing the no. of edges b/w every pair of nodes



$a_{ij} = a_{ji}$

$\forall i, j$

	A	B	C	D
A	0	1	0	0
B	0	0	1	0
C	0	1	0	1
D	0	1	0	0

i

j

Adjacency matrix storing the no. of edges b/w every pair of nodes

Adjacency matrix :

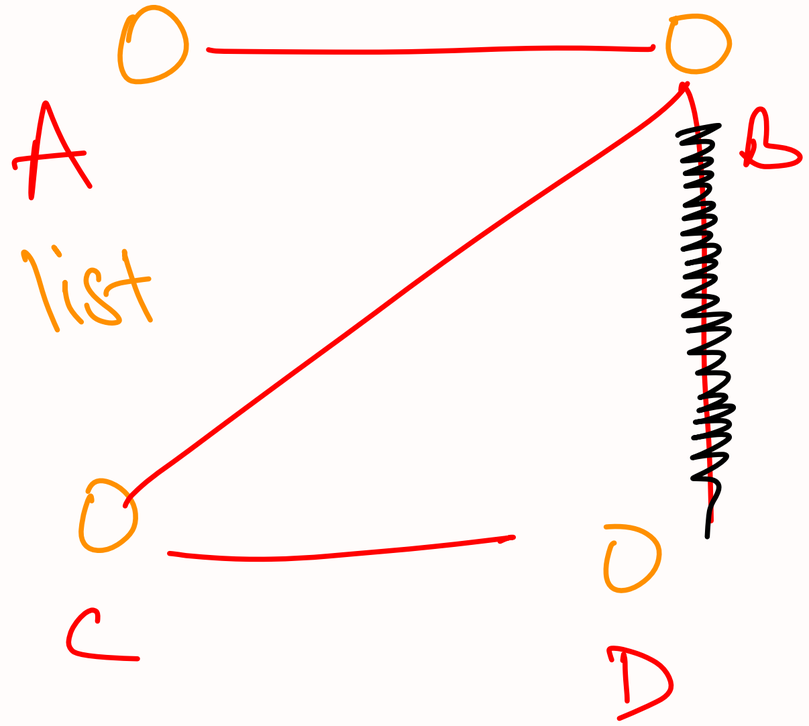
Space $\rightarrow$ $O(n^2)$

Time $\rightarrow$ $O(edges)$

to populate

1. Time to insert an edge $\rightarrow$ $O(1)$
2. Time to delete an edge $\rightarrow$ $O(1)$
3. Time to find out degree of a node $\rightarrow$ $O(n)$

adjacency list with vector



A → [B]

B → [A, C, D]

C → [B, D]

D → [B, C]

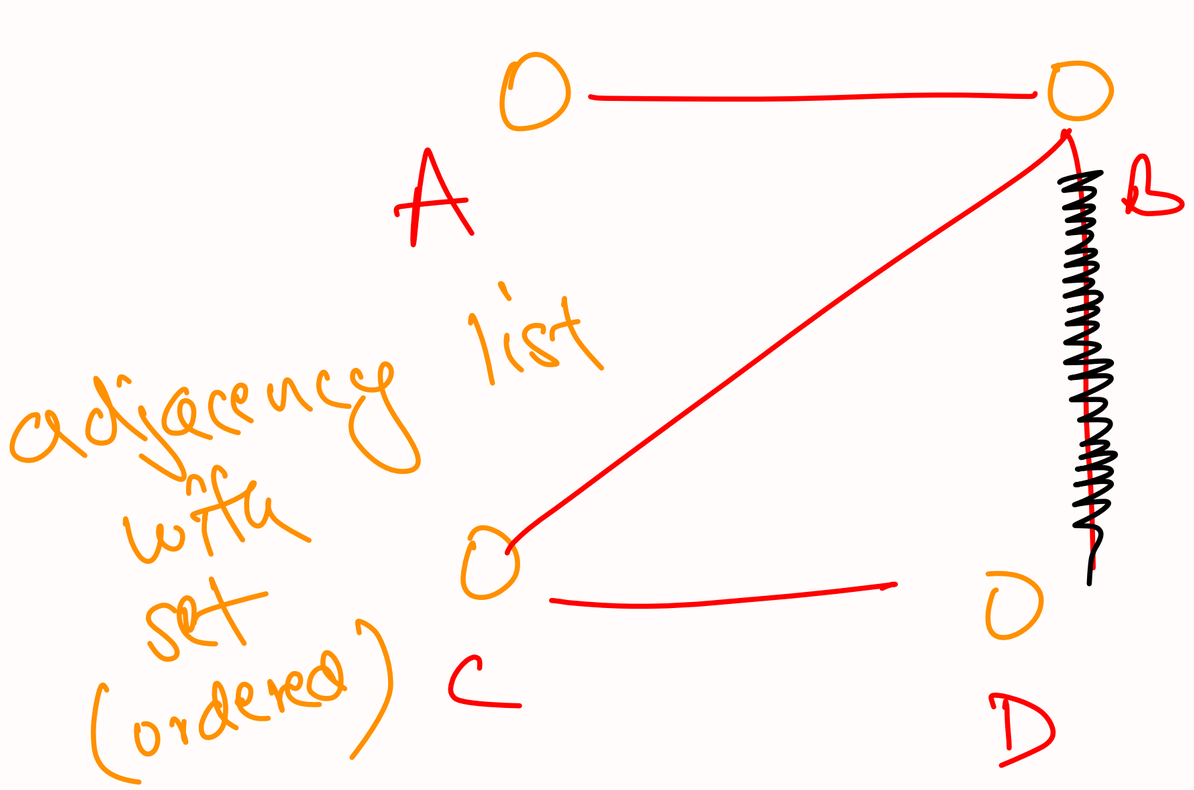
space → O(edges)

time to populate → O(edges)

time to insert an edge → O(1)

time to delete an edge → O(n)

time to find out degree → O(1)



adjacency list with set (ordered)

A → [B]

B → [A, C, D]

C → [B, D]

D → [B, C]

space → O(edges)

time to populate → O(e·logn)

time to insert an edge → O(logn)

time to delete an edge → O(logn)

time to find out degree → O(1)

Trade off	adjacency matrix	adjacency list with vector	adjacency list with set
space	$O(n^2)$	$O(e)$	$O(e)$
time to populate	$O(e)$	$O(e)$	$O(e \log n)$
insert edge	$O(1)$	$O(1)$	$O(\log n)$
delete edge	$O(1)$	$O(n)$	$O(\log n)$
degree of node	$O(n)$	$O(1)$	$O(1)$

$1 \le n \le 1000$, $1 \le e \le 10^7$, $1 \le q \le 10^6$

query → add an edge / delete an edge

finally after all queries → print the degree of each node

matrix → $O(n^2)$ space $O(e + q + n^2)$ time

adj list with vector → $O(e)$ space $O(e + q \cdot n + n)$ time

adj list with set → $O(e)$ space $O(e \log n + q \log n + n)$ time

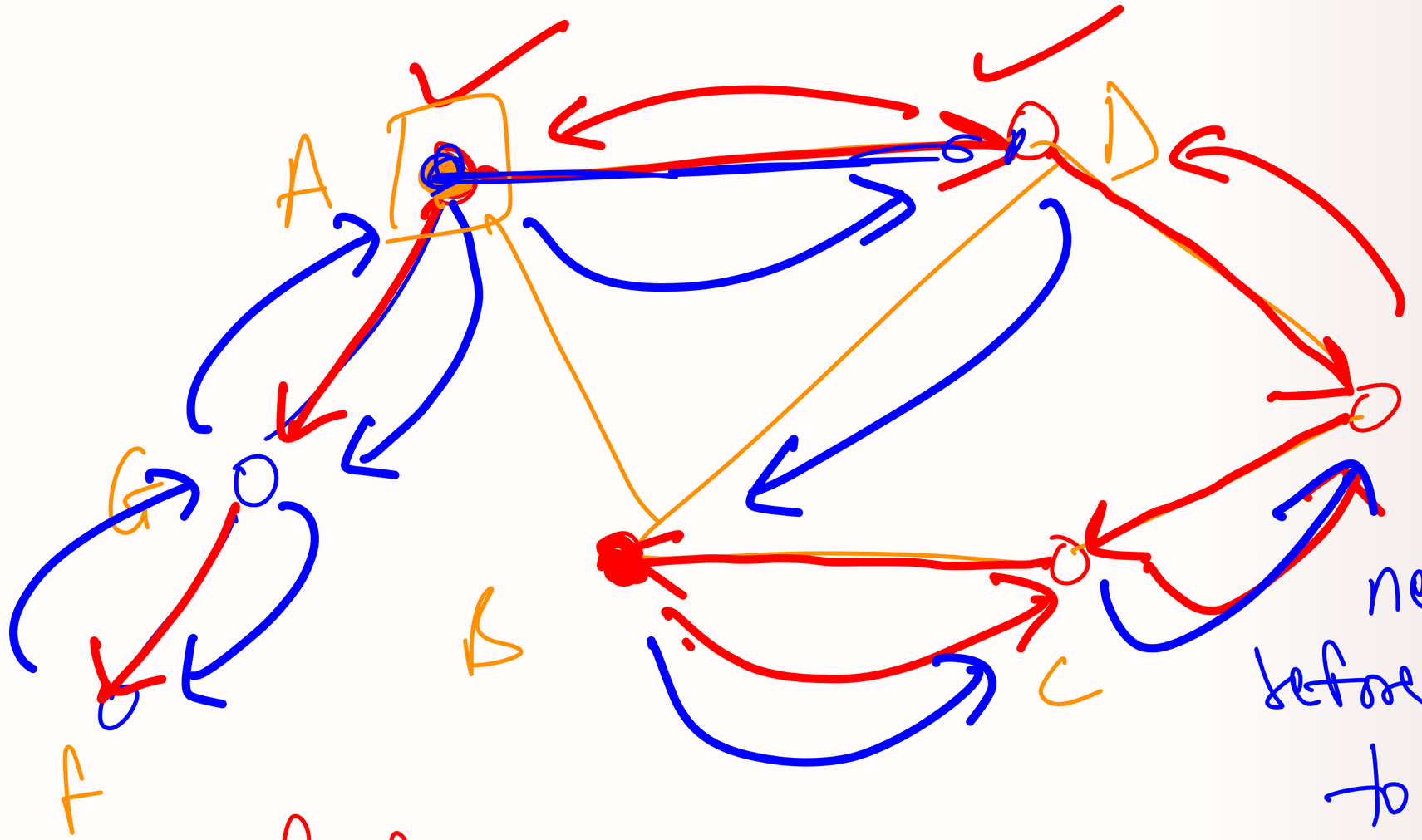
in 99% problems you will be given

$1 \leq n \leq 10^5$, $1 \leq e \leq 10^5$

99.9% problems you won't have to delete edges

Traversals

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

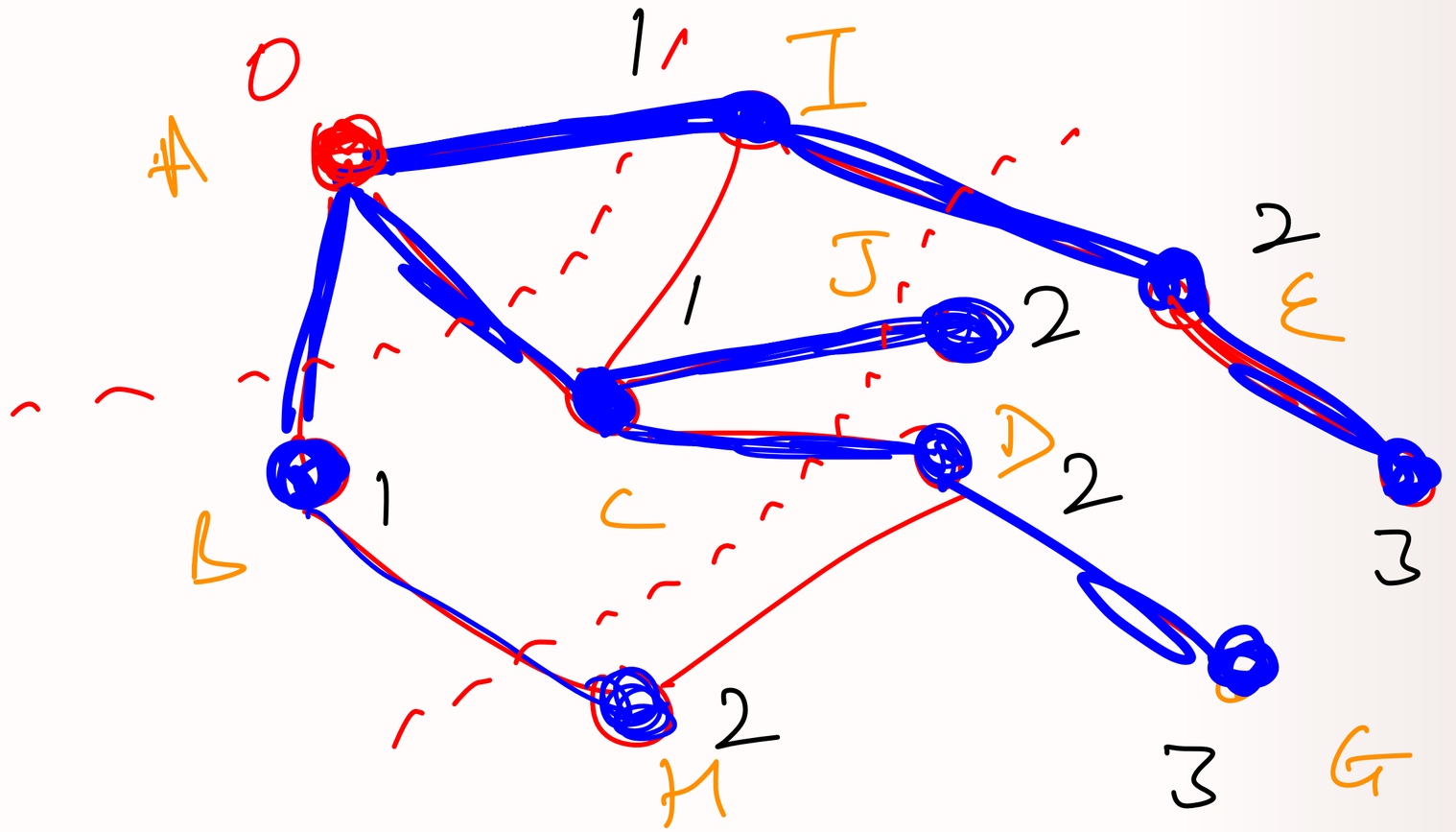


Depth first search

↓

explore every neighbour completely before coming back to the previous node

Red
Blue
} both are valid DFS traversals here



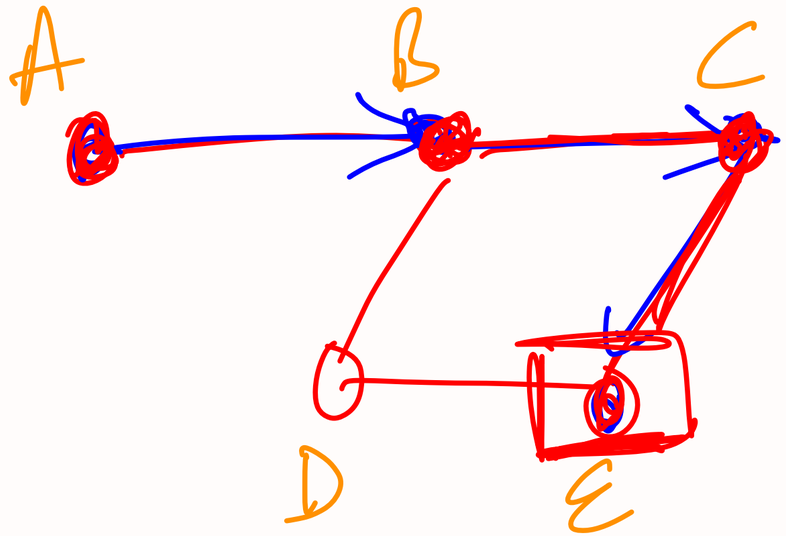
BFS

explore all the neighbours of a node before you go deep into any particular neighbour

no. of connected components

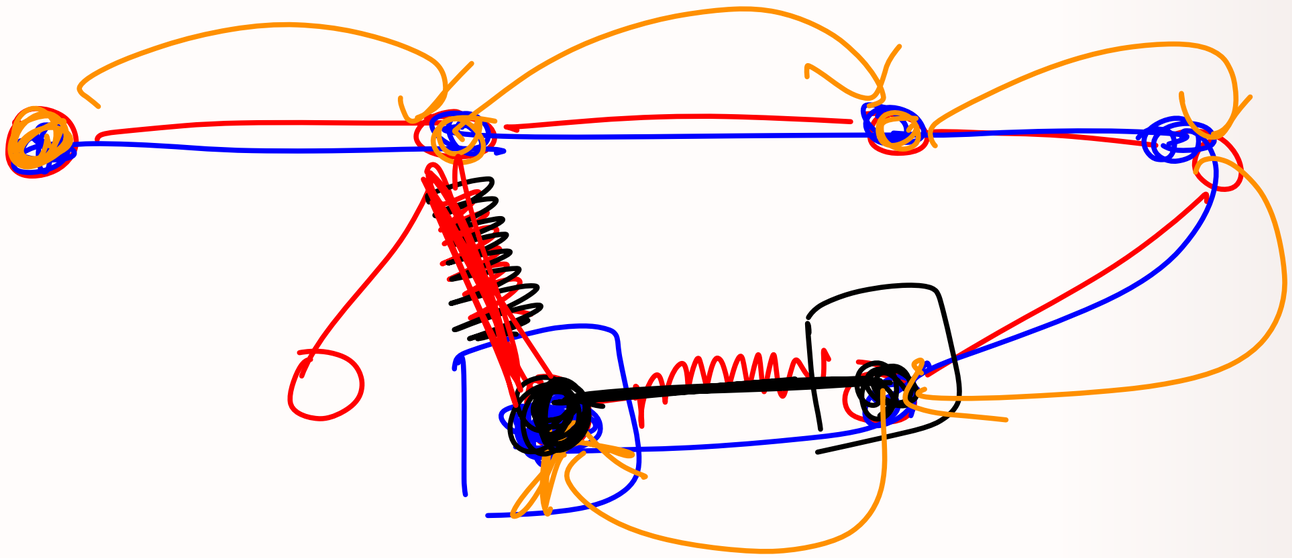
```
vector<bool> vis(n, false)
int count = 0
for (int i = 0 ; i < n ; i++)
        if (!vis[i]){
                dfs(i, edges, vis);
                count++;
        }
}
```

find out any path b/w two nodes

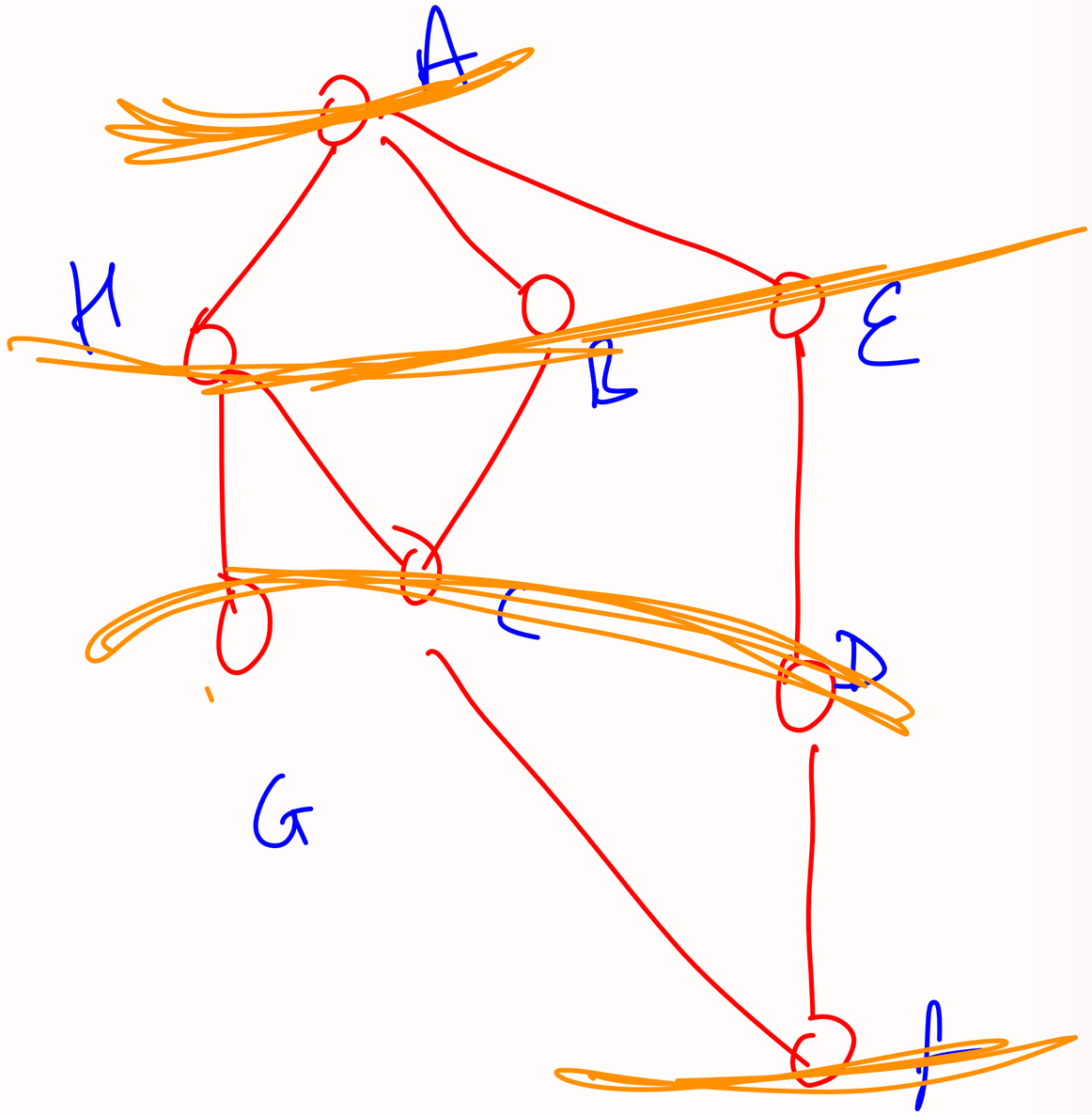



```
dfs(curr, edges, vis, prev, parents){
    vis[curr] = true
    parents[curr] = prev
}

while(curr != -1){ ans.push(curr)
    curr = parents(curr) }  reverse the ans
```



check while doing DFS that for any node all the neighbours (except the parent) must not be already visited for an acyclic graph


A
H
K
E
C
D
G
F