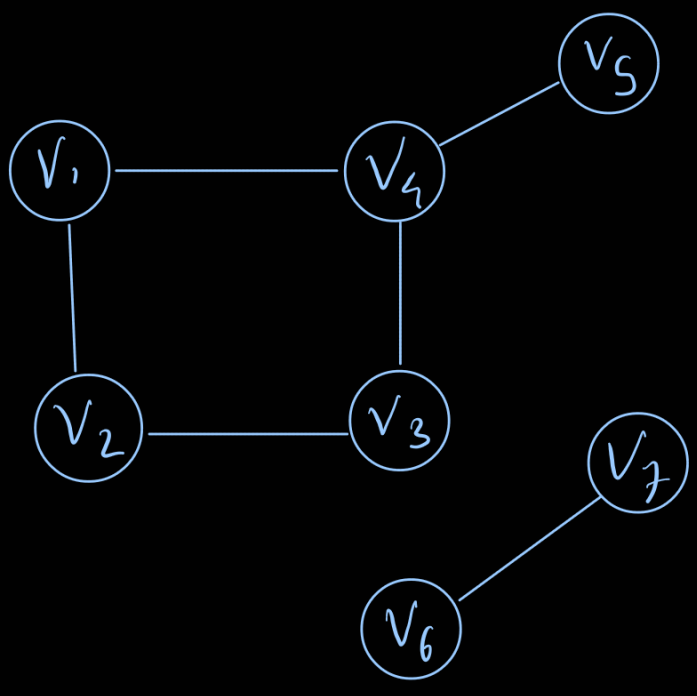

DSA
GRAPHS
SHIVANSH SANTOKI

What is a Graph?

Formal:

→ A graph $G = (V, E)$ consists of a vector set V and edge set E, such that $E \subseteq V \times V$

Eg:



$|V| = 7$

$|E| = 6$

$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$

$E = \{(v_1, v_2), (v_1, v_4), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_6, v_7)\}$

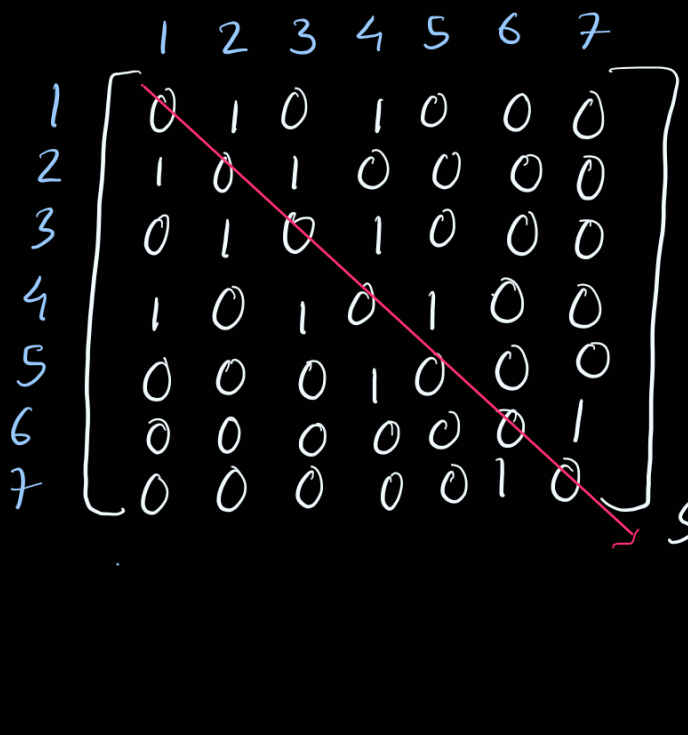
Adjacency Matrices:

Denotes if ∃ a edge b/w two vertices or not.

If edge b/w i and j nodes → $a_{ij} = 1$

else → $a_{ij} = 0$

∴ In this case, the matrix will be 7×7.



→ What is the maximum number of edges that a graph $G = (V, E)$ on n vertices $(|V| = n)$ can have?

$$|E| = {}^{n}C_2 = \boxed{\frac{n(n-1)}{2}}$$

(Edge btw every two nodes)

→ How many different graphs are there on the vertex set $V = \{1, 2, \ldots, n\}$

(${}^{n}C_2$ ways of this)

Pick any two vertices from the set, there are 2 possible cases, either 1 or 0.

$$\therefore (2)^{{}^{n}C_2} = \boxed{(2)^{\frac{n(n-1)}{2}}}$$

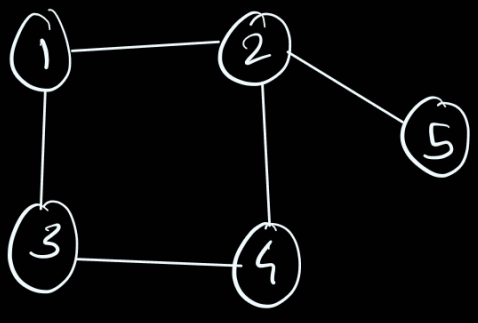
Proof: ${}^{2}C_0 + {}^{2}C_1 + {}^{2}C_2 + \cdots + {}^{2}C_2 = (2)^2 = (2)^{\frac{n(n-1)}{2}}$

↑ zero edges ↑ one edge

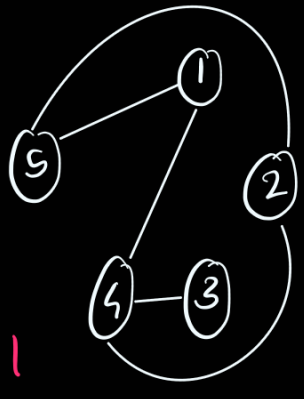
Matrix

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

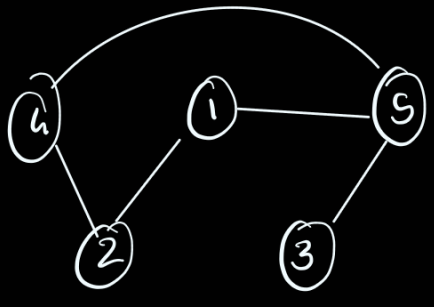
Graph



Eg:

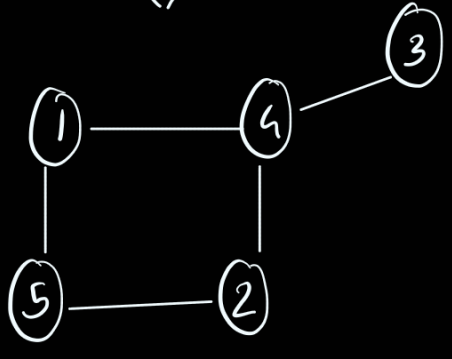


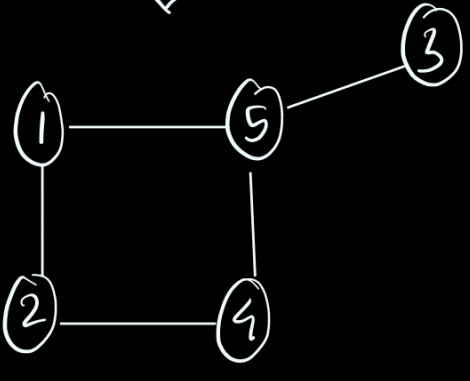
G1



G2

G1 and G2 are isomorphic





Two graphs are isomorphic to each other if the rows & colums of the adjacency matrix of one of them can be exchanged so that these are same as the adjacency matrix of other.

NOTE:

Given two graphs G_1 and G_2, both with n vertices, check if they are isomorphic or not.

$TC(\text{Best Algo}) = \boxed{2^{O(\log n)^3}}$ (Babai, 2016)

✗ 3 Ways of Representing Graph:

(A) Adjacency Matrix [nxn matrix, 2D array]

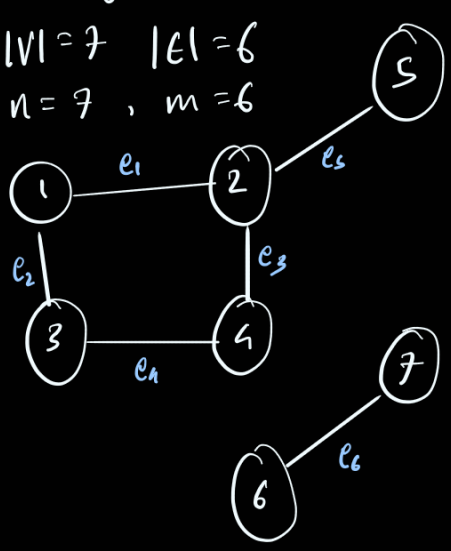
(B) Incidence Matrix [nxn matrix, 2D array]

(C) Adjacency List [n-linked lists, Space: 2n]

let the graph be:

$|V| = 7 \quad |E| = 6$

$n = 7, \quad m = 6$



(A) Adjacency Matrix:

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

If $(v,u) \in E$

then,

$Adj[v][u] = 1$

Space = $n \times n = 7 \times 7$

(B) Incidence matrix:

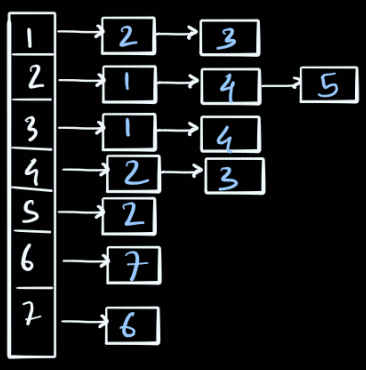
	e_1	e_2	e_3	e_4	e_5	e_6
1	1	1	0	0	0	0
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	0	0	1	1	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1
7	0	0	0	0	0	1

Space:
$n \times m$
$= 7 \times 6$

Disadvantage:
If $n = 20,000$, then 19,998 elements in each column will be zero.

$m \leq \frac{n(n-1)}{2}$

(C) Adjacency List:



Space: $2E = 2 \times 6 = 12$

	Adjacency List	Adjacency Matrix
Space Complexity	$O(2m)$	$O(n^2)$
$(v,u) \in E$?	$O(n)$	$O(1)$
All neighbours of a vertex	$O(n)$	$O(n)$
All edges	$O(n+m)$	$O(n^2)$
Insert edge	$O(1)$	$O(1)$
Delete edge	$O(n)$	$O(1)$

Neighbour:
A vertex v is said to be a neighbour of u, if $(v,u) \in E$

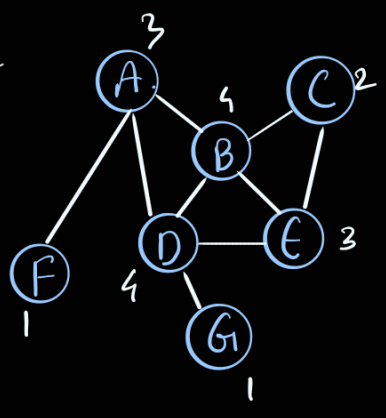
Degree:
The degree of a vertex is the number of neighbours it has.

$$0 \leq deg(v) \leq n-1$$

Handshaking Lemma:

If each person at a party shakes hands with those that the person is friends with, then the total number of handshakes is equal to half the [(cummulative total of) the number of handshakes each person made].

Eg:



$$\deg(A) + \deg(B) + \cdots + \deg(G)$$
$$= 3 + 4 + 2 + 4 + 3 + 1 + 1$$
$$= \boxed{18}$$

$n(\text{Edges}) = \boxed{9}$

$$\therefore \boxed{n(E) = \frac{\Sigma \deg(\forall V)}{2}}$$

Fact: No. of people, who shakes odd number of hands is even.

M-1

Let $n = |V|$

Proof: and, $m = |E|$

As each edge contribute to the degrees of two vertices,

$\Sigma \deg(V_i) = 2m$

Let x = no. of vertices with odd degree.

$\therefore \Sigma \deg = (n-x)(\text{even}) + (x)(\text{odd})$

$\therefore 2m = \text{even} + x \times \text{odd}$

$\therefore \text{even} = \text{even} + x \times \text{odd}$

$\therefore \text{even} = x \times \text{odd}$

$\therefore \boxed{x = \text{even}}$

M-2 Empty Graph: 0 is even

There are 3 cases:

Case (1): (A) odd (B) odd ⇒ (A) even — (B) even

Case (2): (A) even (B) even ⇒

Case (3): (A) odd (B) even ⇒

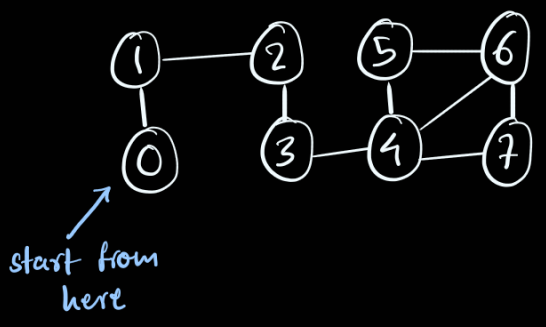
※ Breadth - First - Search (BFS):

- Start with a vertex A.
- Then explore the neighbours of A.
- Repeat this for each of the neighbours of A.
  And so on...
- Keep exploring until there is nothing left to explore.

Pseudo Code:

```
BFS(A)
{
    Push (null, A) to queue Q.
    while(Q is nonempty)
    {
        pop(p, v) from Q
        if(visited(v) == FALSE)
        {
            visited(v) ← TRUE
            parent(v) ← p
            for each edge (v, w)
                push (v, w) to Q
        }
    }
}
```

Eg: Let |V| = 8, |E| = 9



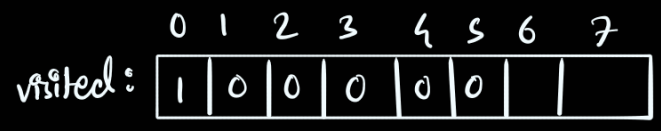
visited:

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

BFS Traversal : ans

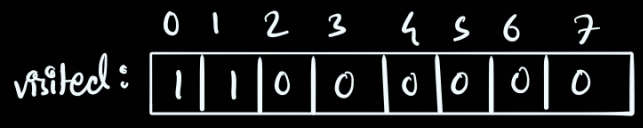
1) queue : (null, 0) pop

ans : 0



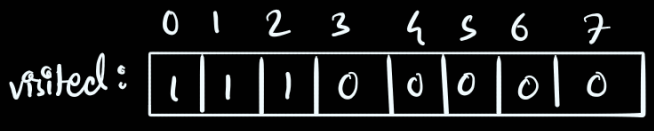
2) queue : (0, 1) pop

ans : 0 1



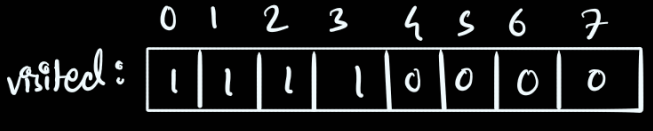
3) queue : (1, 2) pop

ans : 0 1 2



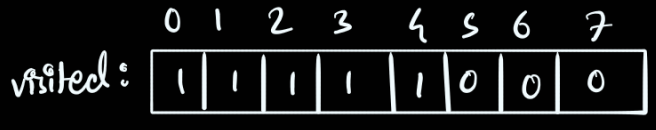
4) queue : (2, 3) pop

ans : 0 1 2 3



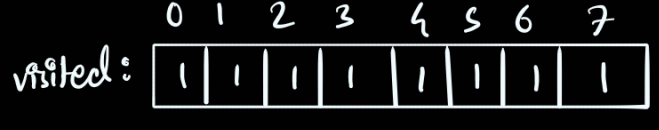
5) queue : (3, 4) pop

ans : 0 1 2 3 4



6) queue : (4, 5), (4, 6), (4, 7) pop

ans : 0 1 2 3 4 5

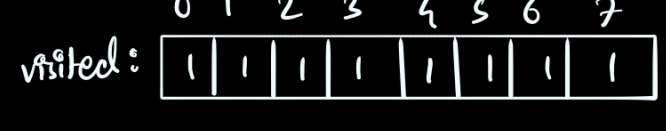


7) queue : (4, 6), (4, 7) pop

ans : 0 1 2 3 4 5 6

8) queue : (4, 7) pop

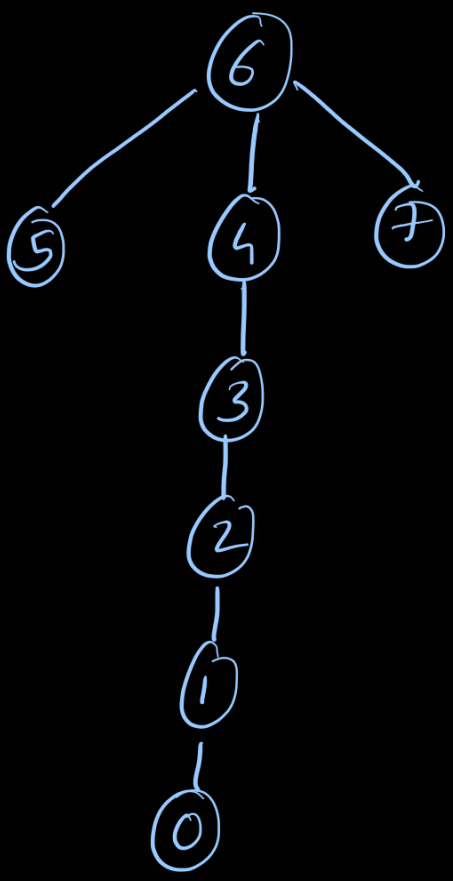
ans : 0 1 2 3 4 5 6 7

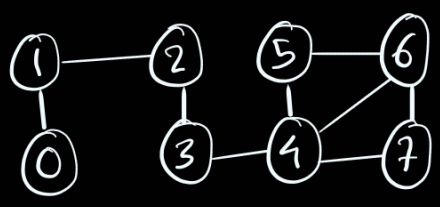


Q empty → loop terminates.

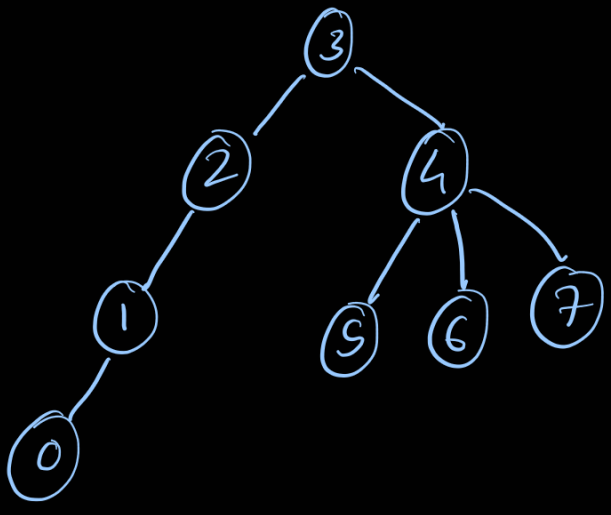
Q: What will be the BFS tree for

(i) BreadthFirstSearch(6):

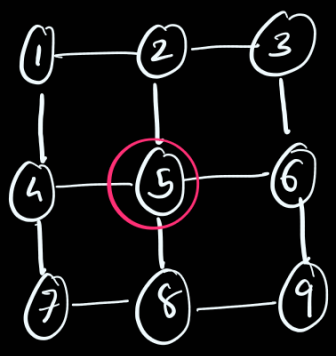




(ii) BreadthFirstSearch(3):



Q: Given graph:



$|V| = 9$

$|E| = 12$

What is a forward edge:

An edge from vertex to its parents

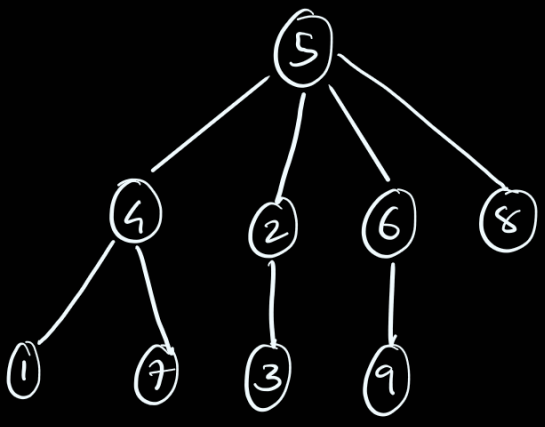
Cross edge:

Edge b/w two vertices at the same level

Back edge:

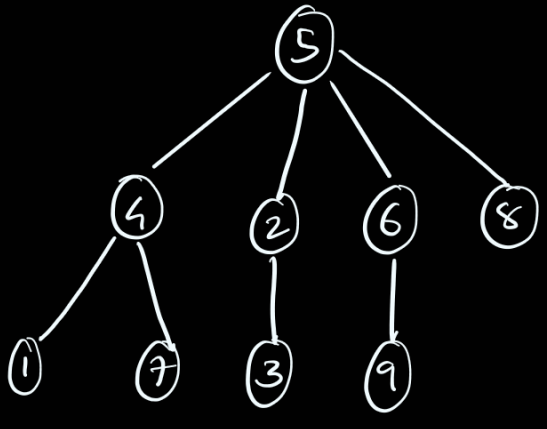
Edge from a vertex at a higher level to a vertex at a lower level.

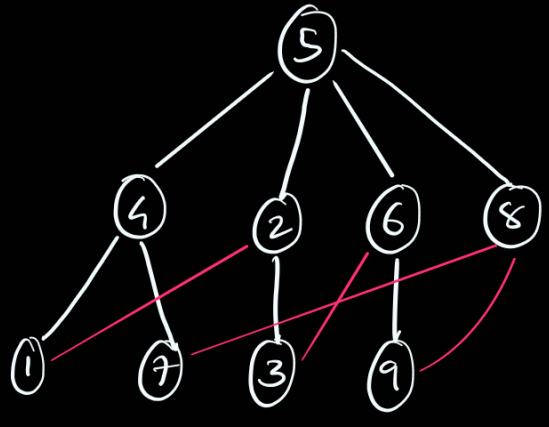
BFS Tree:



forward edges = 8

(f)





Cross edges : 0

(c)

back edge : 4

(b)

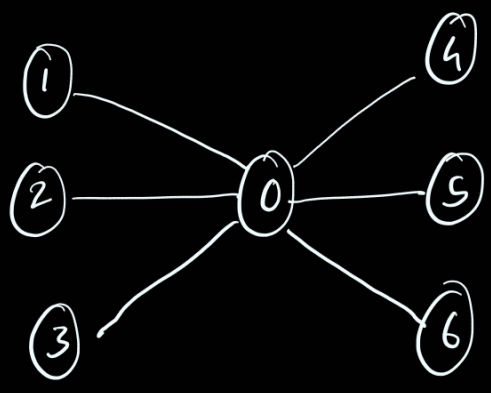
$f + c + b = |E|$ ✓

$8 + 0 + 4 = 12$

→ A graph which does not have any cross edges is a bipartite graph.

→ No odd length cycle.

Eg: Given Tree:

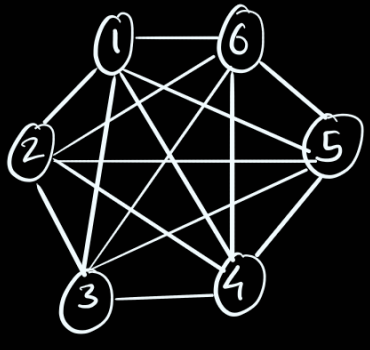


Here, every edge is a forward edge.

A graph which has 0 cross and back edges is a tree. (All edges are forward edges)

$|V| = 7 \quad |E| = 6$

Eg: Complete Graphs:



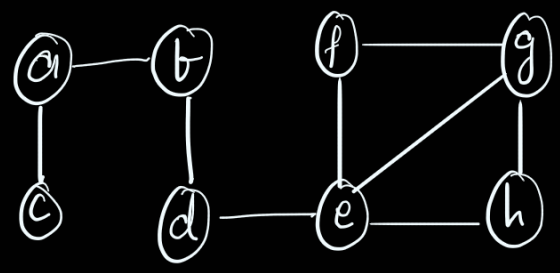
※ Depth First Search

Pseudo Code:

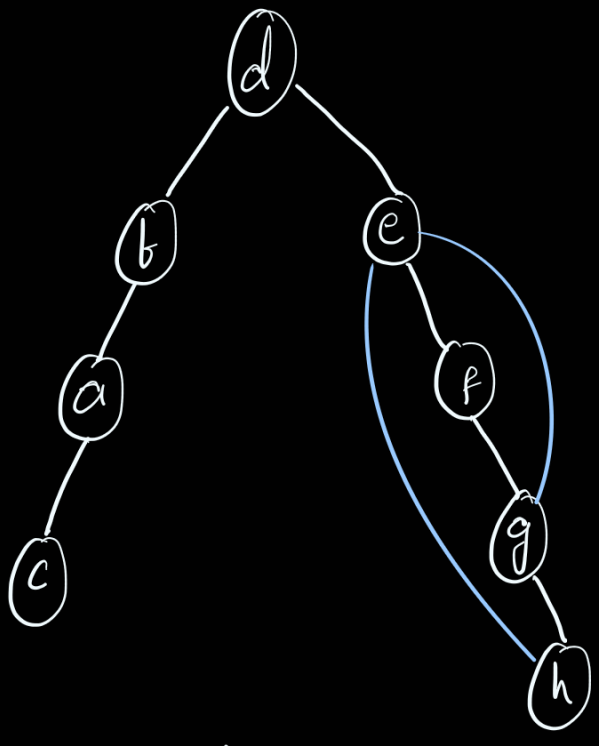
```
DFS(s)
{
    push(null, s) to stack S
    while(S is not empty)
    {
        pop(p, v) from S
        if(visited(v) == FALSE)
        {
            visited(v) ← TRUE
            parent(v) ← p
            for(each (v, w) ∈ E) {
                push(v, w) to S
            }
        }
    }
}
```

→ Look at a neighbour of S.
→ Then, look at a neighbour of neighbour of S.
→ Keep traversing in this way and terminate when everything is traversed.

Eg: Consider the following graph:



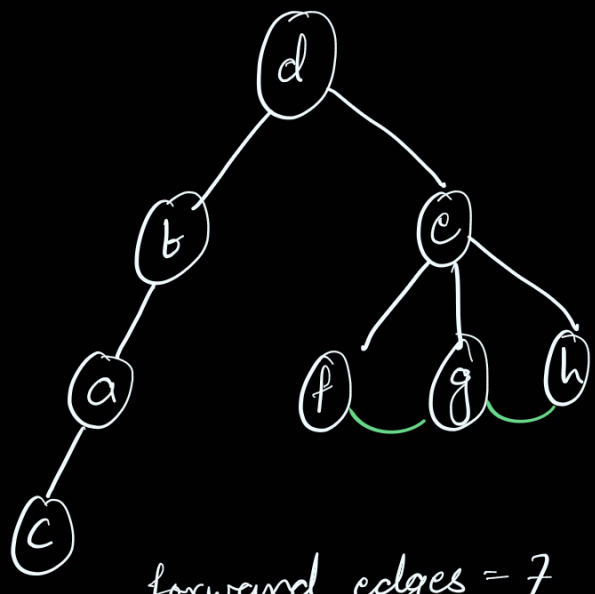
DFS(d)



forward = 7

back edges = 2

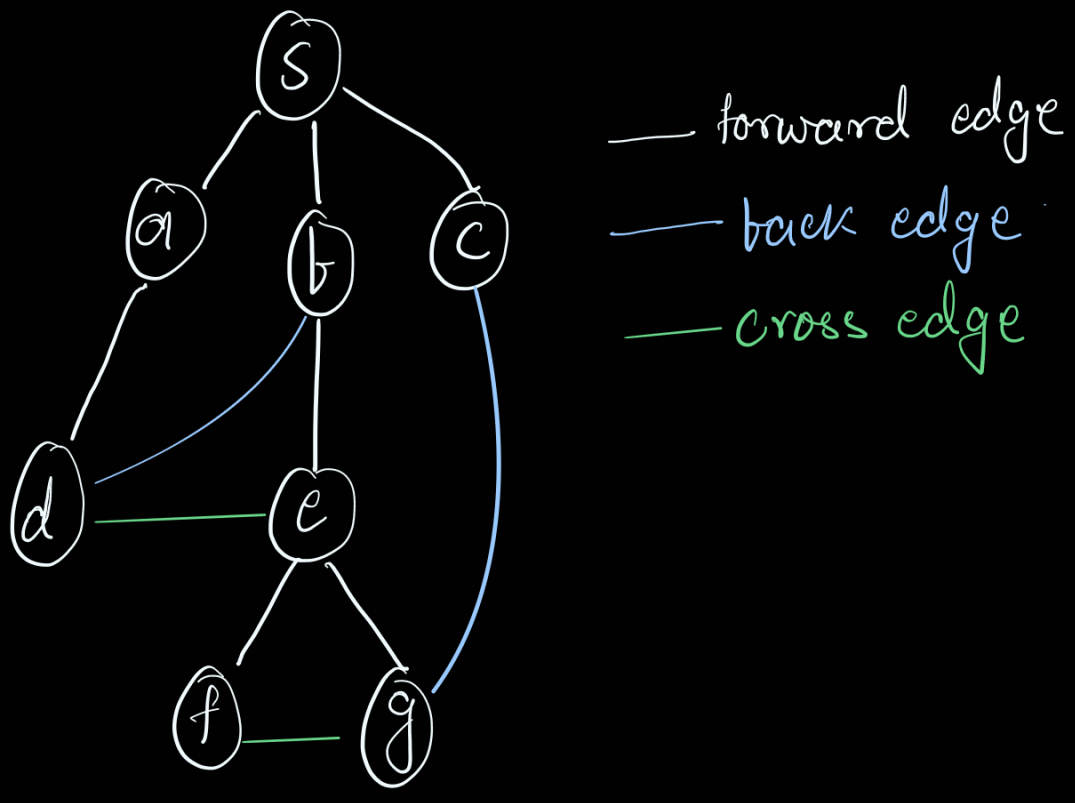
BFS(d)



forward edges = 7

cross edges = 2

Q. Construct the graph for which this is a BSF traversal. You may add more edge if you want.



Facts:

1) Let v be a vertex in level i of a BFS traversal of a graph.
Then, all its neighbours are in one of the level: {i-1, i, i+1} i.e

2) If ∃ an edge b/w vertex at level i and level j, then $|i-j| \leq 1$.

Hence, the given BFS traversal is wrong.

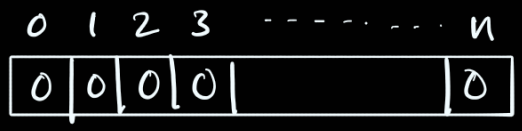
NOTE: The level of a vertex in its BFS traversal from S is it's distance from s. (only true for BFS).

Another Fact:

In $O(|V|+|E|)$, it is possible to compute distances of all vertices from s.

Proof: Do BFS traversal. Let $|V|=n$, $|E|=m$

let Dist :



Let $s=1$,

then the array "Dist" will store the distance of i from 1.

→ Initially push(1,0)

[vertex, level]

then traverse all of its !visited neighbours:

for each neighbour (k) of (s, level):

```
if !visited(k)
    Dist[k] = level + 1
    push(k, level + 1)
    visited(k) = 1
```

※ Types of Graph :

Type	Definition
① Empty Graph	A graph with no edges
② Complete Graph	A graph in which all pairs of vertices are edges
③ Path	A chain of connected vertices without repetition
④ Cycle	A loop of connected vertices without repetition
⑤ Tree	∃ exactly one path b/w every pair of vertices.
⑥ Bipartite Graph →	A graph whose vertices can be partitioned into two sets, L & R such that, every edge has one end-point in L and one end-point in R.
⑦ Regular Graph	
⑧ Planar Graph	

NOTE: While doing BFS traversal on a Bipartite graph, any two vertices at the adjacent level are in different sets.

A graph is a Bipartite graph iff it has no cross edge.

※ Bipartite Graph:

→ A graph is bipartite iff its BFS traversal has no cross-edge.

→ A graph is bipartite iff it has no odd-length cycle.

※ Cycles in BFS:

→ Odd cycles ⟺ cross edge

→ Even cycle ⟺ back edge

※ Trees

→ No odd cycles

→ No even cycles

→ Only forward edges.

→ Every tree is a bipartite graph.

※ Disconnected Graphs:

→ A graph is disconnected iff it contains two vertices u and v such that there is no path between u and v.

→ The empty graph on $n \geq 2$ vertices is disconnected.

※ Graph Isomorphism:

→ Two graphs are isomorphic if one of them can be relabeled such that their adjacency matrix are identical.

→ To show that graphs are NOT isomorphic, show one property of one graph that does not exist in the other graph.

Undirected Graphs

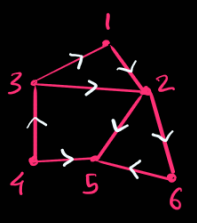
- $G = (V, E)$
- $E \subseteq V \times V$
- $(v, v) \notin E$
- $(u, v) \in E \iff (v, u) \in E$
- $A(i)(j) = A(j)(i)$

  [Adjacency matrix is symmetric]

Directed Graphs

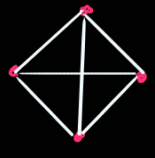
- $G = (V, E)$
- $E \subseteq V \times V$
- $(v, v) \notin E$

Eg:

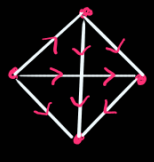


	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	0	0	1	1
3	1	1	0	0	0	0
4	0	0	1	0	1	0
5	0	0	0	0	0	0
6	0	0	0	0	1	0

Q Consider the following Graph:



(1) Put arrows on the 8-edges such that the resulting graph is acyclic.



(2) An undirected graph such that such it is impossible to convert is into a "acyclic" directed graph.

∄ any such graph.

Proof: → Given any undirected graph, put a arrow such that it goes from a larger number to a smaller number.

→ As $a > b > c \cdots > a$ is not possible, the resultant directed graph will be acyclic.

※ Shortest Path:

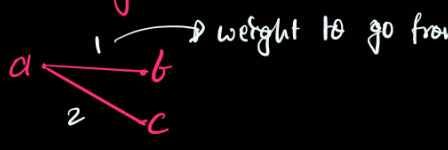
* Single source shorted path (SSSP):

→ Given a directed graph G and a vertex S of G (source), compute shorted path from s to every other vertex of G.

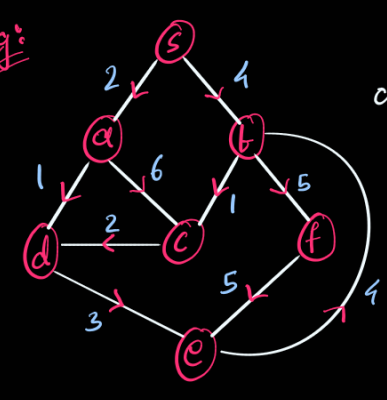
Eg:

→ Run the BFS from s.
The level number for each vertex denotes the distance of that vertex from the source.

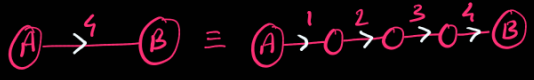
* Weighted edges:



Eg:



dist(s,c) = min(2+6, 4+1)
= 5



→ Modify the graph in the following way, so that we do not need to change the algorithm.