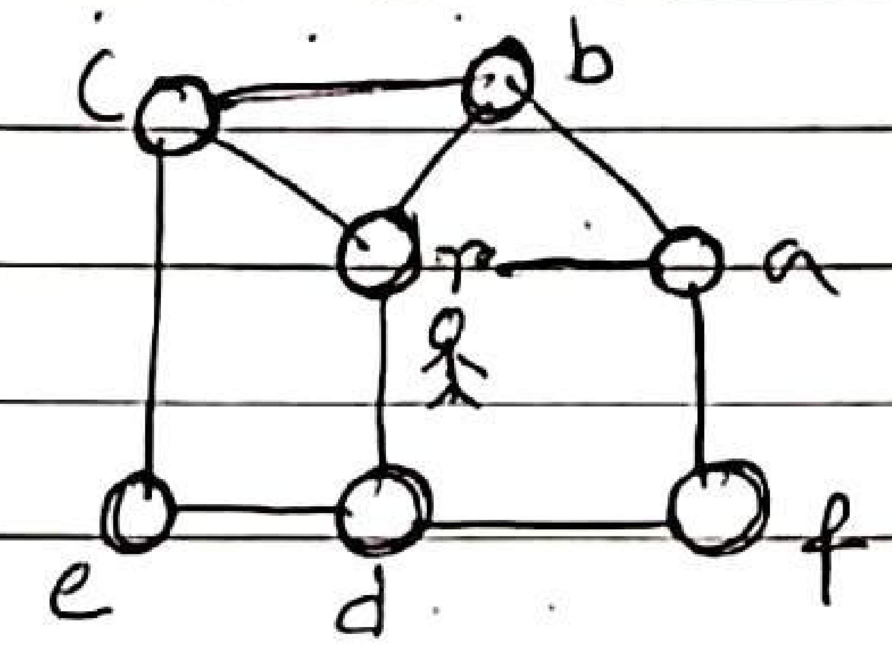
GRAPHS

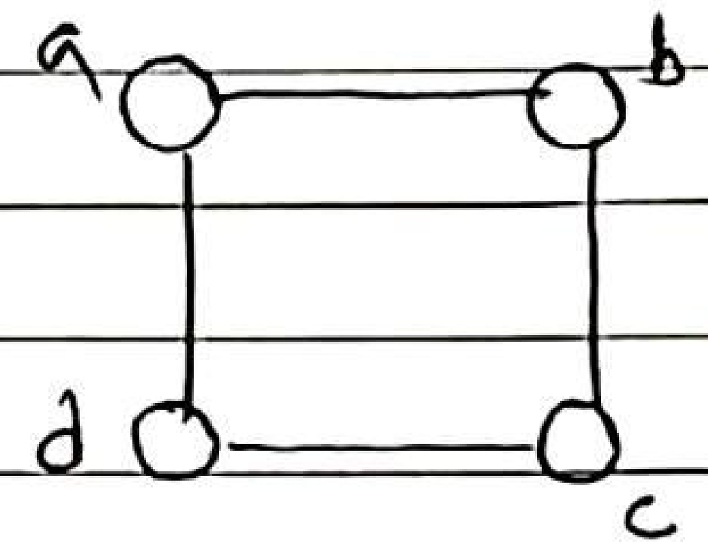
- Introduction of Graphs =

→ set of vertices and edges is called graph.



(friend)
- adjacency of x - (dcba)
(friend list)

- Two popular representation of graph =
  - → adjacency matrix.
  - → adjacency list.

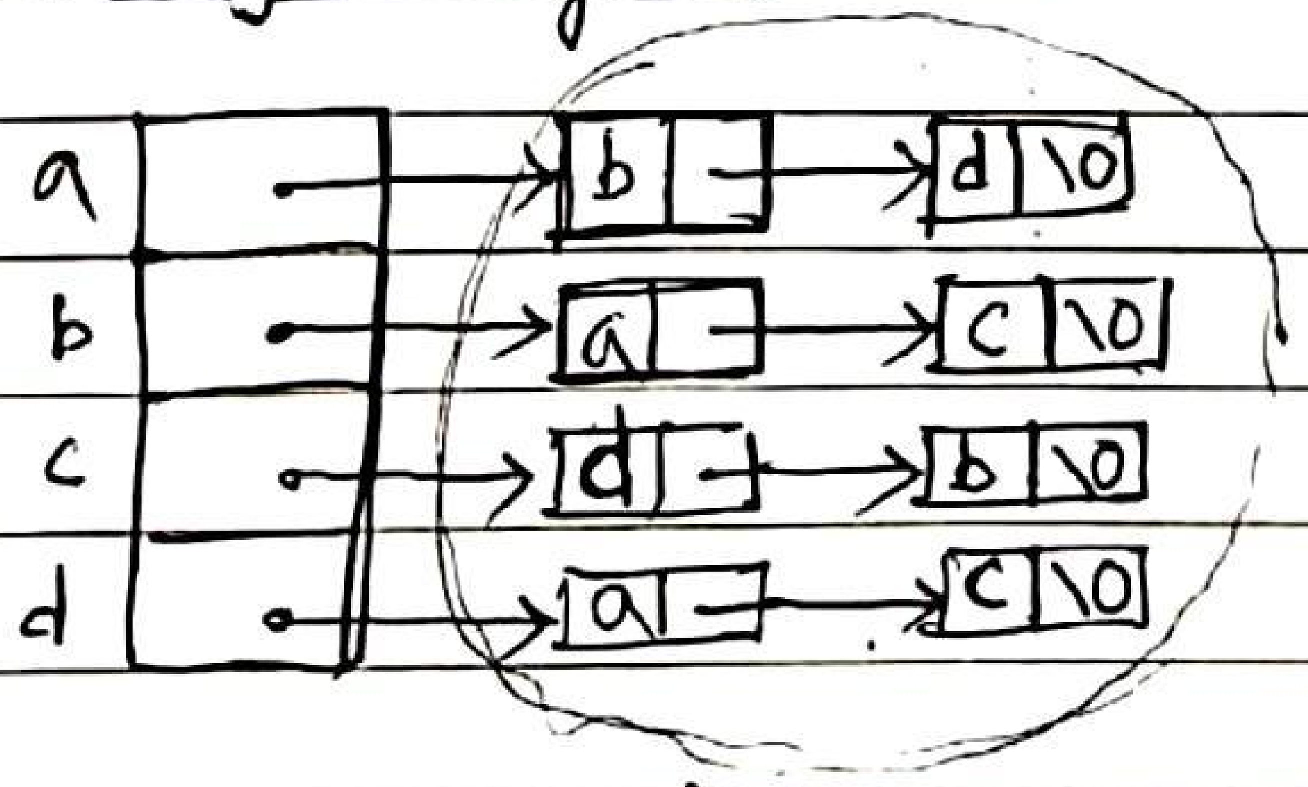


- Adjacency matrix =

	a	b	c	d
a	0	1	0	1
b	1	0	1	0
c	0	1	0	1
d	1	0	1	0

space required -
$O(v^2)$

- ~~Link list~~ Adjacency list =



space required -
$O(v+2E)$
$= O(v+E)$.

→ When the graph is ~~Dese~~ Dense (if the no. of edge very very high) then go with matrix representation.

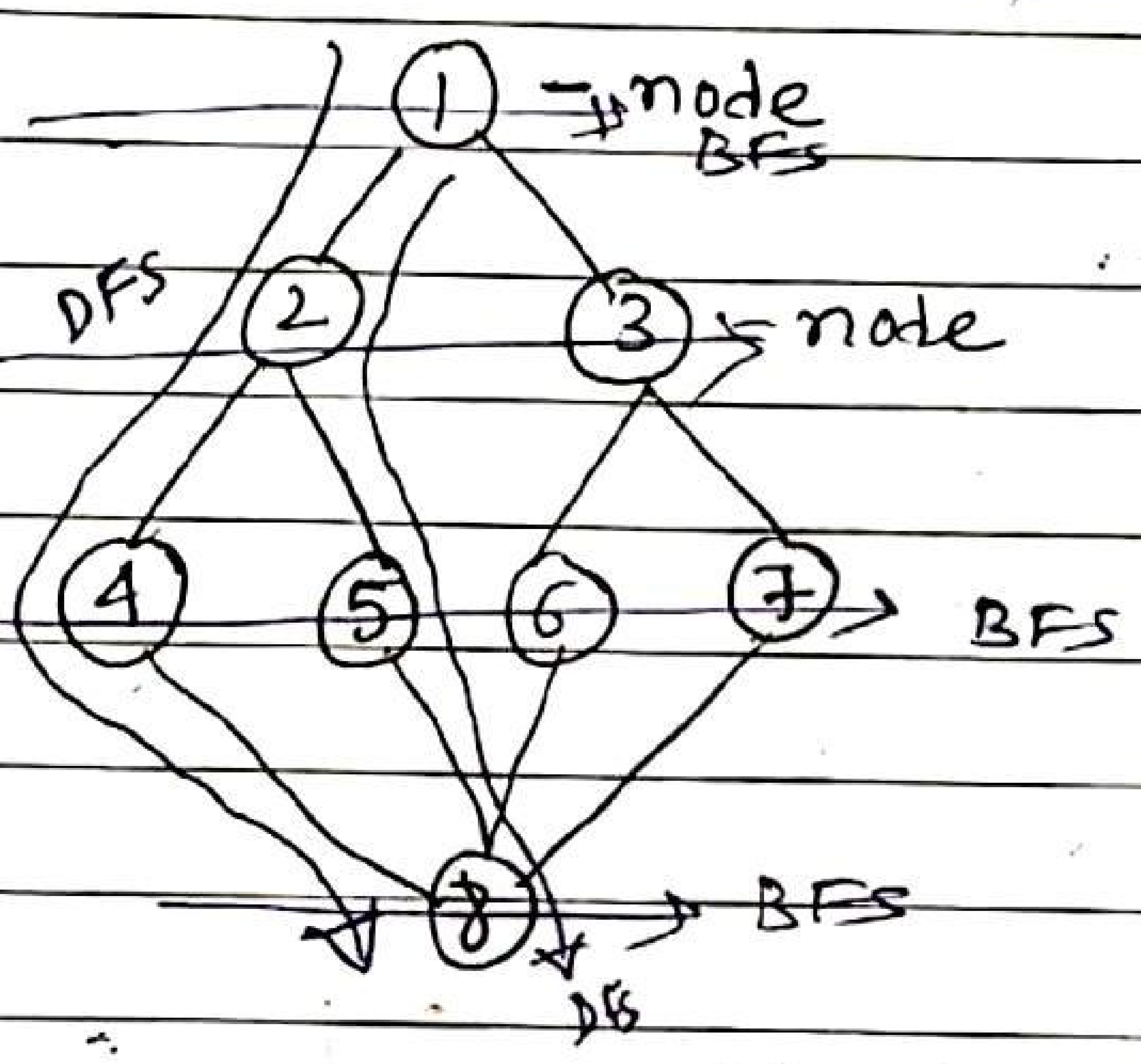
$E = O(v^2)$

→ When the graph is sparse (few edges), then go with ~~edo~~ Adjacency list representation.

$$E = O(V).$$

- **Introduction of BFS and DFS** =

BFS	DFS
↓	↓
breadth first search	depth first search



search all the node in the given graph by using Breadth first search and depth first search.

vertices 3-types —
↳ visited, not-visited, Explored.

visited	Explored
↓	↓
seen that vertex	seen it and seen all vertex adj with that vertex.

	VISITED	EXPLORED	
Case-1	0	0	→ not visited, not explored.
C-2	~~0~~ 1	0	→ visited, but not explored.
C-3	1	1	→ visited and explored.

→ Keep track of this two things (~~visited, Explored~~ visited or not) both the algorithm maintain an array-

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

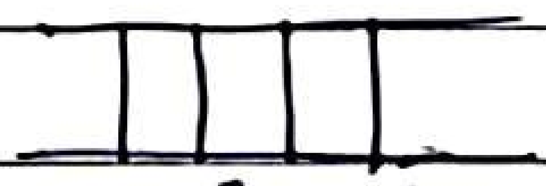
VISITED

1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0

visited — not visited.

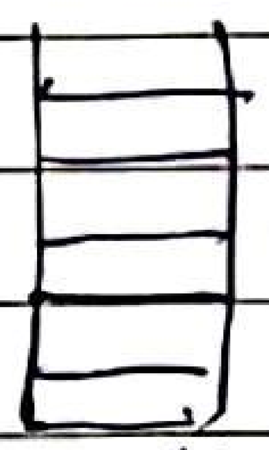
✓ space com = O(n).

→ The Algo which use the queue to keep track of all unexplored vertice is called - BFS.



Queue — BFS ✓ space complexity = O(n)

→ The algo which use stack to keep track of all unexplord nodes is called - DFS.



stack — DFS

✓ space complexity = O(n).

- **BFS algorithm** =

BFS(v) → address of 1st node

// The graph 'G' and array visited[] are global; visited[] is initialized to '0'.

```
{
   u = v ;   visited[v] = 1;
   repeat
   {
      for all vertices w adj to u
      { if (visited [w] == 0)
         { add w to queue;
           visited [w] = 1; }
      }
```

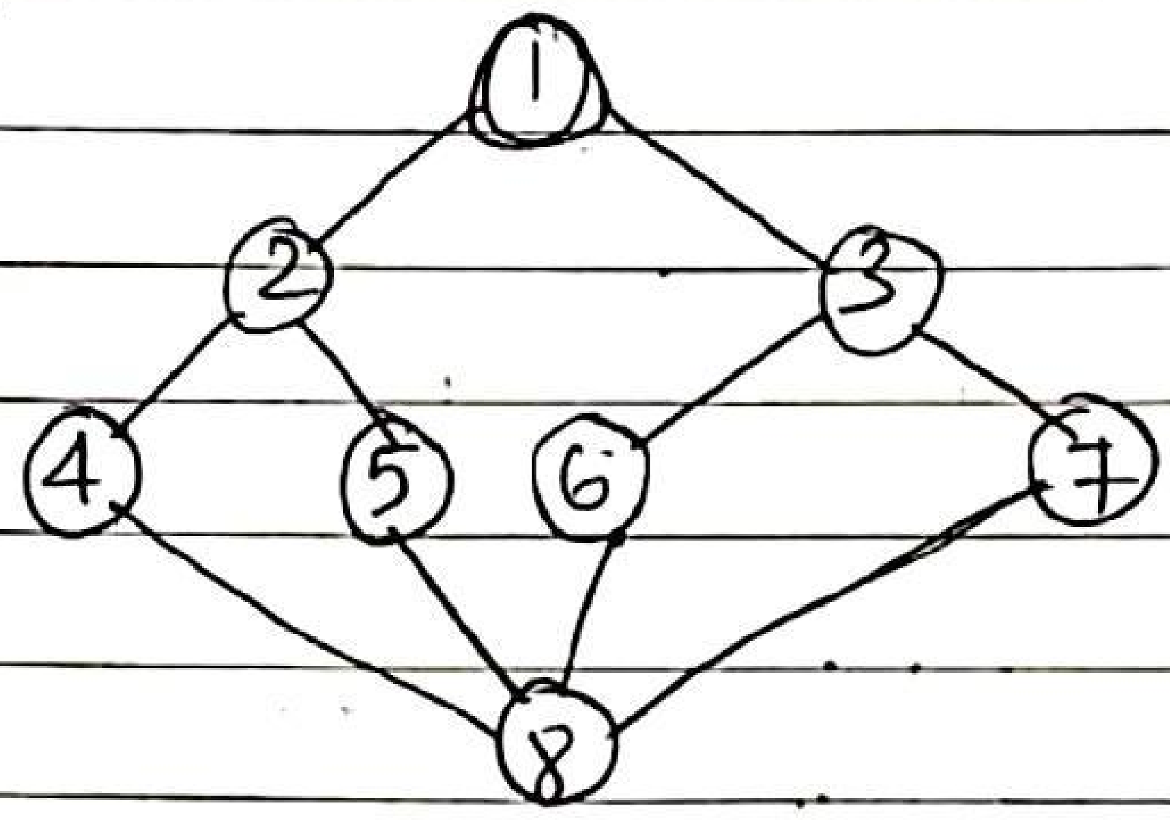
→ if ('w').

if queue is empty then return;
Delete the next element, from queue, and add to u;

}

}

example =



	1	2	3	4	5	6	7	8
array	~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ 1

u = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ 8 ↳ NOW all vertices visited

~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~	~~8~~

queue

v = 1
w = 2, 3 ✓

v = 2
w = 1, 4, 5 ✓

v = 3
w = 1, 6, 7 ✓

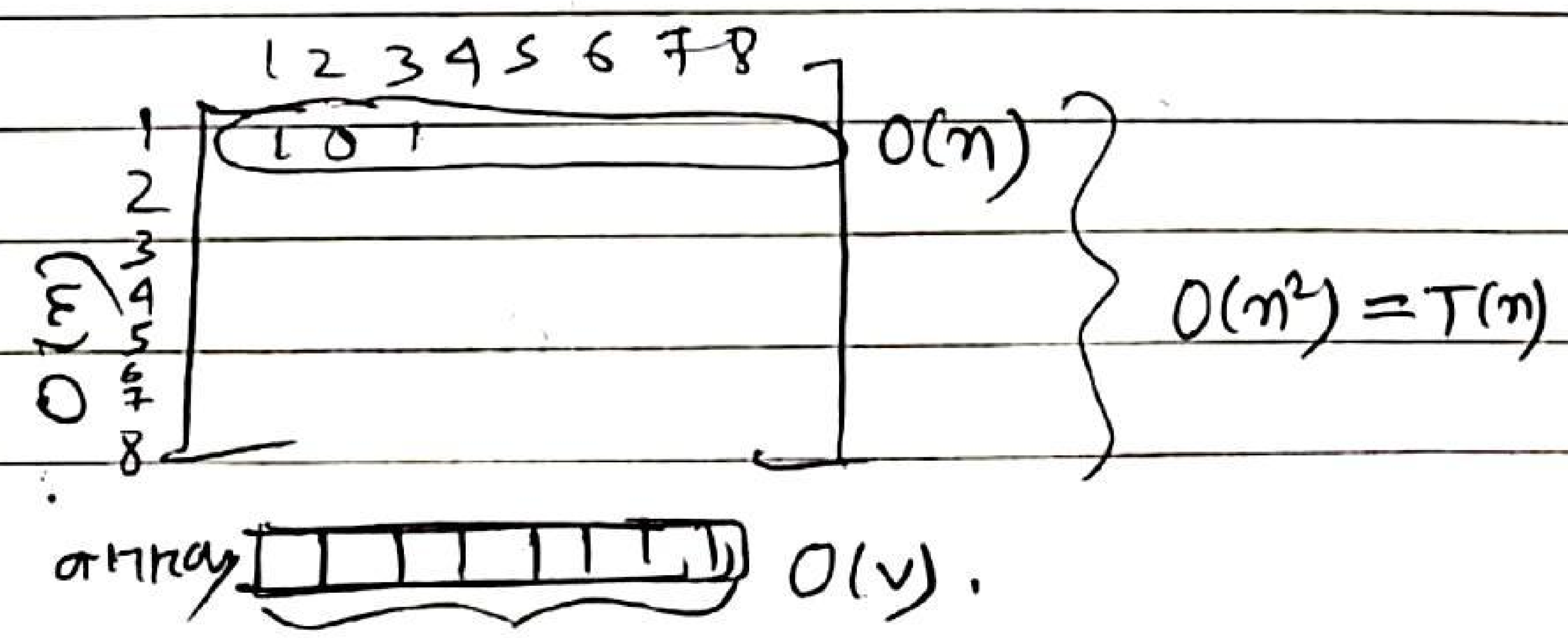
v = 4
w = 2, 8

• $\boxed{\text{BFS}}$ analysis on adjacency matrix implementation =

Time complexity $T(n) = O(n^2)$
$= O(v^2)$

$\boxed{v \to \text{vertices}}$

space Complexity $= O(v)$

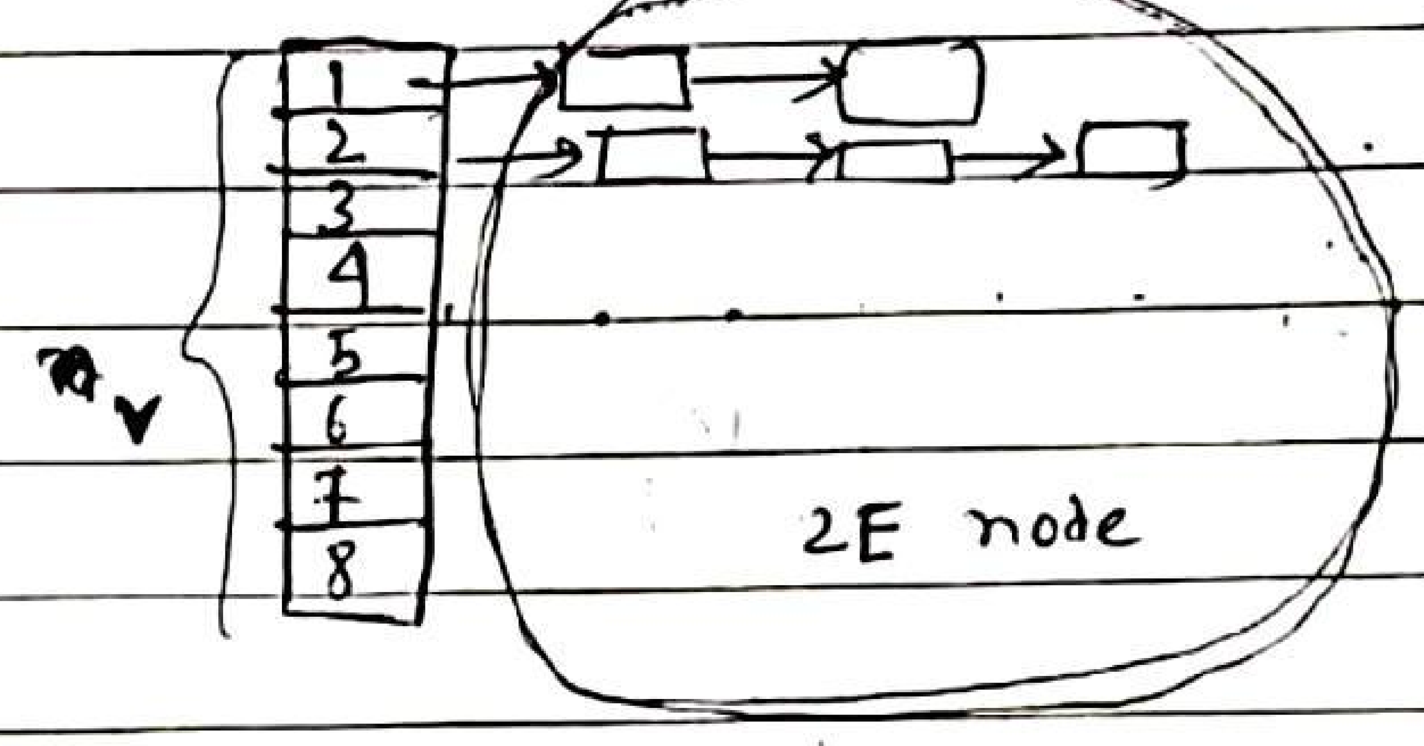


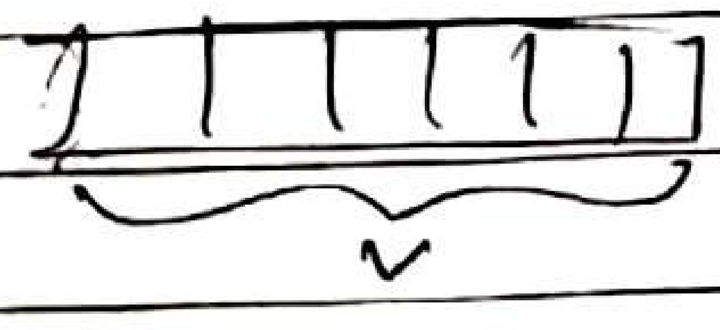
- **BFS analysis in case of linked list implementation of Graph =**

✓ Space complexity in Worst case $= O(n)$
$= O(N)$

✓ time complexity $= O(V+E)$

E → edges

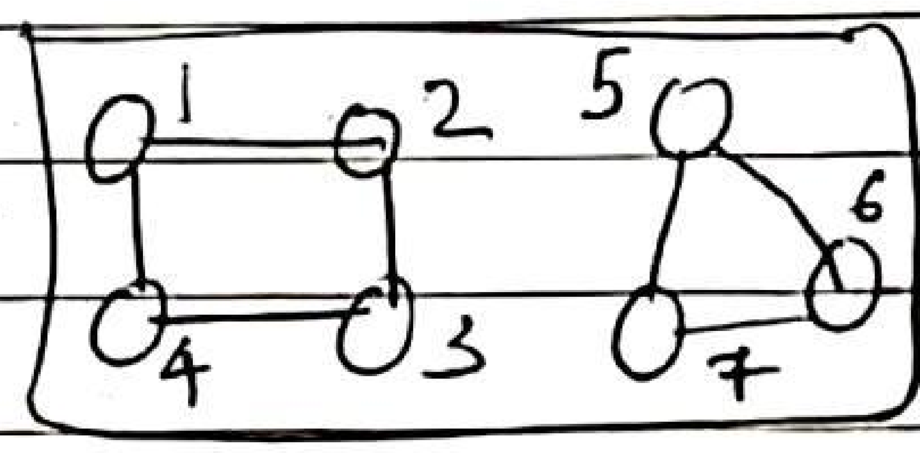




- **Breadth First Traversal using BFS =**

```
BFT (G, n)
{
    for i=1 to n do
        visited[i] = 0
    for i=1 to n do
        if (visited[i]==0) then
                        BFS(i);
}
```

i =	1	2	3	4	5	6	7	8
visited	0	0	0	0	0	0	0	0
	1	1	1	1				



~~Time complexity = $O(n^2)$ $O(E+V)$~~
~~Space complexity =~~

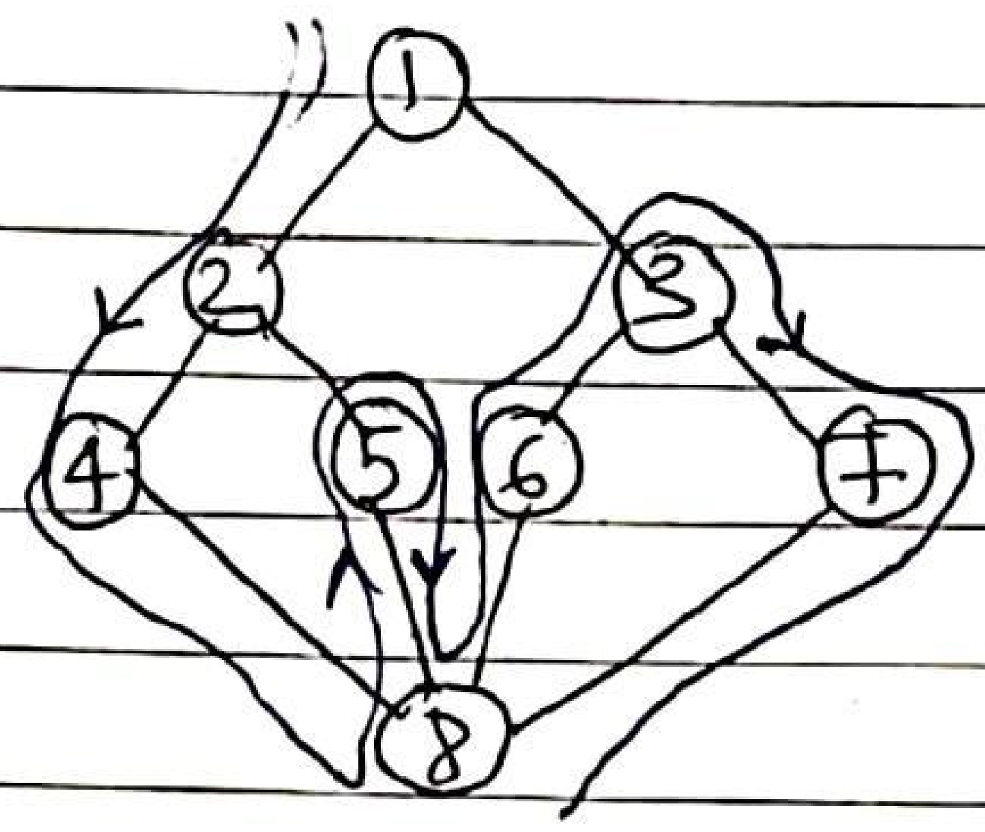
→ Time and ~~com~~ space complexity of BFT are same as BFS.

- **DFS algorithm =**

```
DFS(v)
{
  visited[v] = 1;

  for each vertex w adj to v do
    { if (visited[w]==0) then
        DFS(w);
    }
}
```

example –



DFS.

visited

1	2	3	4	5	6	7	8
~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
1	1		1	1	1	1	1

stack

~~V=1 W={2,3}~~	~~V=2 W={1,4,5}~~	~~V=4 W={2,8}~~	~~V=8 W={4,5,6,7}~~	~~V=5 W={2,8}~~	~~V=6 W={3,8}~~	~~V=3 W={1,6,7}~~	~~V=7 W={3,8}~~

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

- Analysis of DFS and DFT =

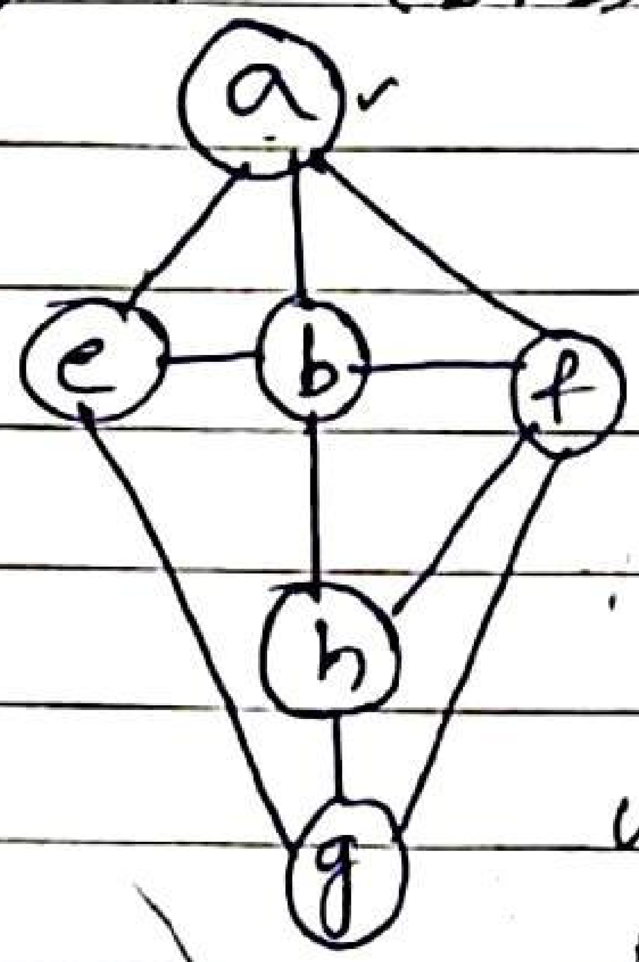
→ Incase of Adj Matrix –
Time complexity = $O(V^2)$
Space complexity = $O(V)$

→ Incase of Adj list –
Time complexity = $O(V+E)$
Space complexity = $O(V)$.

→ Space and Time complexity are same in both the case of DFS and DF Traversal.

g-2003 $\boxed{\text{Question}}$ - 1

(DFS) - depth first search -



✓ I. abeghf. (possible)

× II. abfehg. (not possible)

✓ III. abfhge. (possible)

✓ IV. afghbe. (possible)

Which of the following sequence ~~poss~~ not possible?

→

V=a	~~V=e~~	~~V=b~~
W={e,b,f}	~~W={a b g}~~	

I.

Ex

V=a	V=b	V=e	V=g	~~V=h~~	~~V=f~~
W={e b f}	~~W={a e f h}~~	W={a b g}	~~W={e h f}~~	~~W={b f g}~~	~~W={a b h g}~~

abeghf

II & III.

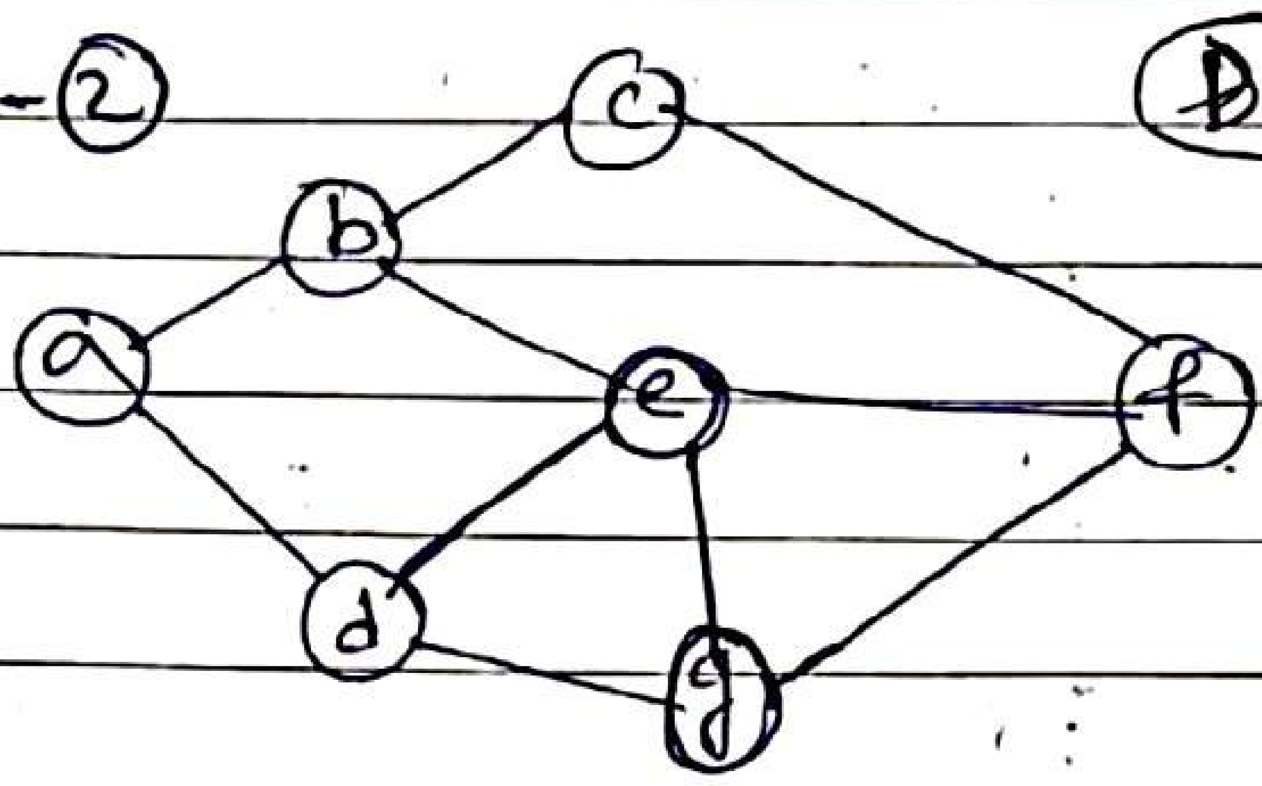
V=a	V=b	V=f	V=h	V=g	V=e
W={e b f}	W={a e f h}	W={a b h g}	W={b f g}	W={e h f}	W={a b g}

IV.

V=a	V=f	V=g	V=h	V=b	V=e
W={e b f}	W={a b h g}	W={e h f}	W={b f g}	W={a e f h}	W={a b g}

gate-2008

Question - 2 (DFS)



Which of the following sequence are possible —

1) abefdgc. (not possible)
2) abefcgd.
3) adgebcf.
4) adbcgef. (not possible)

⟹

1), 2)

v = a	v = b	v = e	v = f	v = c	v = g
W = {b, d}	W = {a, c, e}	W = {b, d, f}	W = {c, e, g}	W = {b, f}	W = {d, f, e}

3)

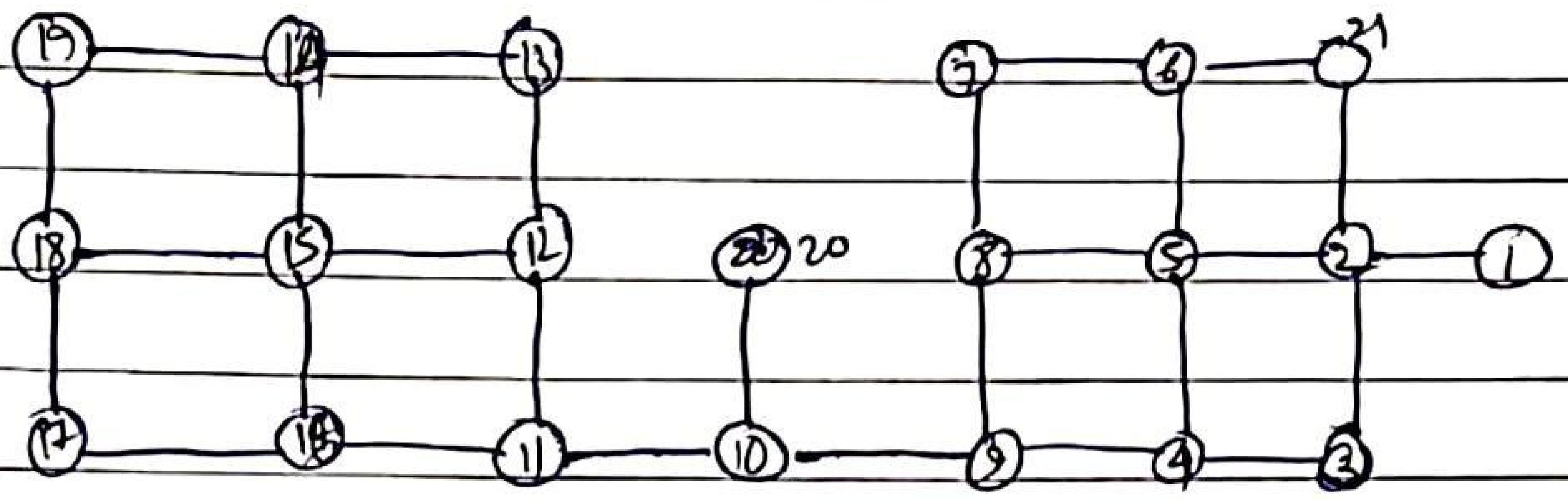
v = a	v = d	v = g	v = e	v = b	v = c
W = b, d	W = {a, e, g}	W = {d, f, e}	W = {b, d, f}	W = {a, c, e}	W = {b, f}

4)

v = a	v = d
W = {b, d}	W = {a, e, g}

g-2014

Question - 3 (DFS)



start visiting from any node -, and find out ~~how~~ how many max node will ~~present at worst~~ at a time in stack.

→

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

stack

(19 node can present at same time on the stack)

gate 2006

Question - 4

consider a DFS of an undirected graph with 3 vertices P, Q, R. let discovery time d(u) represent the time instant when the vertex 'u' is first visited, and finish time f(u) represent the time instance when the vertex 'u' ~~is~~ is last visited Given that -

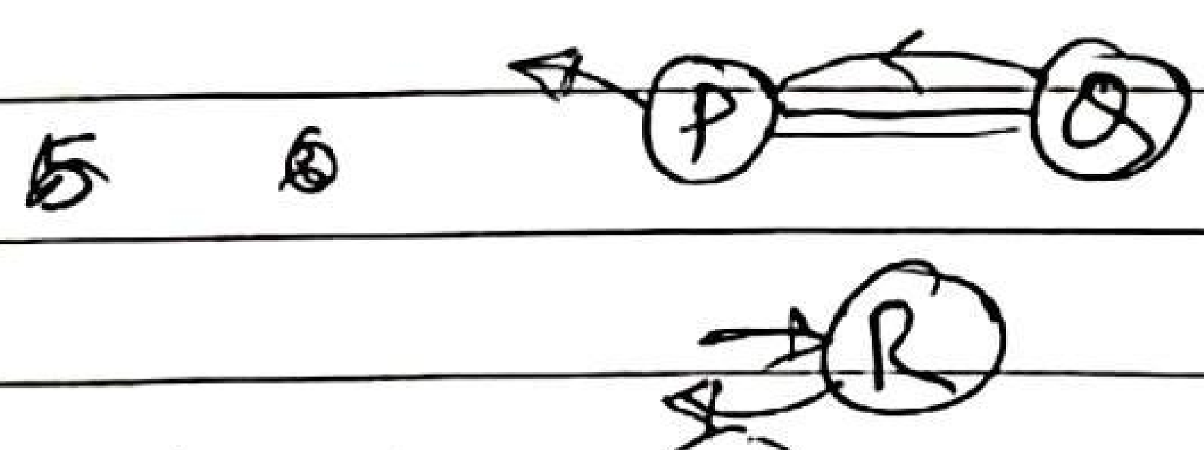
d (P) = 5 Units f(P) = 12 Units

d (Q) = 6 Units f(Q) = 10 Units

d (R) = 14 Units f(R) = 18 Units

What is true about the graph?

→



→ P and Q are in same module.

→ R in other module.

5	6	10	12	14	18
d(P)	d(Q)	f(Q)	f(P)	d(R)	f(R)

→ So, P and Q are adj