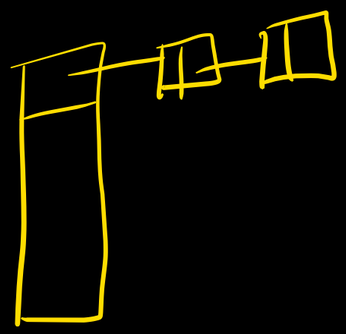
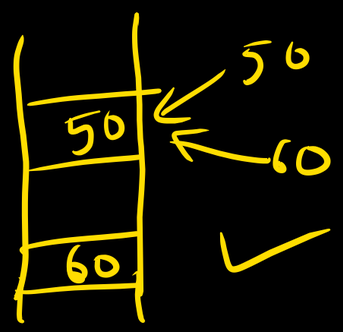
Collision resolution techniques

- Chaining
  - (linked list)
  - (outside)
  - (external hashing)



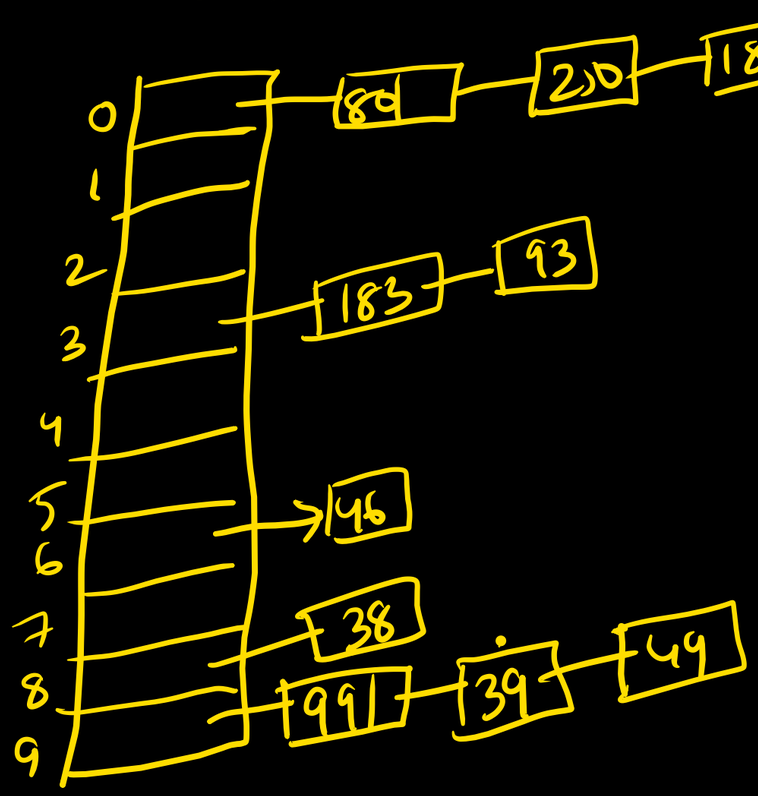
- Open addressing Technique
  - (inside)
  - (internal hashing)



- linear probing
- quadratic probing
- double hashing

Chaining: HT [10] 0-9 m=10 HF: Kmod m

Keys: 80, 99, 46, 39, 38, 250, 180, 183, 93, 49, 66, 35, 120, 190, 60, 89, 69, 39, 62



Collisions : 13

maximum chain length = 7

min chain length = 0

Avg chain length = $\frac{20}{10} = 2$

Disadv :-

→ It takes more space for linked list.

→ In worst case it may generated n length chain.

Searching = $T(n) = O(n)$ [wc]

BC = $T(C) = O(1)$

Insertion $O(1)$

$= O(n)$ if duplicates are not allowed.

Deletion = O(n) (wc)

= O(1) (BC)

Advantages of chaining:-

It can handle ∞ collisions

- m slots → n keys

1 slot → $\frac{n}{m}$ keys/slot ⇒ load factor(α) = $\frac{n}{m}$

$0 \leq \alpha \leq \infty$ (chaining) ✓

$0 \leq \alpha \leq 1$ open addressing

Open addressing :-

Linear probing :-

Ex: $m = 10$ (0-9) Keys: 55, 99, 61, 89, 78, 74, 38, 56, 84

$HF(K) = K \bmod m = K \bmod 10$; CRT = Linear probing

$LP(key, i) = (HF(key) + i) \bmod m$

i ↑ Collision number

$LP(55, 0) = (HF[55] + 0) \bmod 10$

$= (5 + 0) \bmod 10 = 5.$

$LP(99, 0) = 9$

$LP(61, 0) = 1$

$LP(89, 0) = 9 \rightarrow$ Collision

$LP(89, 1) = (9 + 1) \bmod 10$

0	89
1	61
2	38
✓ 3	
4	74
5	55
6	56
7	84
8	78
9	99

LP

↓

LP(K, 0)

→ Collision

↓

go to next cell

~~Disadvanta~~ Disadvantage of open addrs:

If table size is m, then we can put only 'm' elements.

Sometime even if the space is available, we cannot put elements.

Quadratic probing:

m = 10 (0-9)

Key: 55, 99, 61, 89, 78, 74, 38, 56, 84, 69

HF(K) = K mod m ; CRT = QP

$QP(key, i) = \left(\frac{HF(key)}{\text{mod } 10} + \frac{i^2}{=}\right) \frac{\text{mod } m}{\text{mod } 10}$.

QP(69,0) = 9

$= 9 + 1^2 = 0$ $9 + 9^2 = 0$

$= 9 + 2^2 = 3$

$9 + 3^2 = 8$

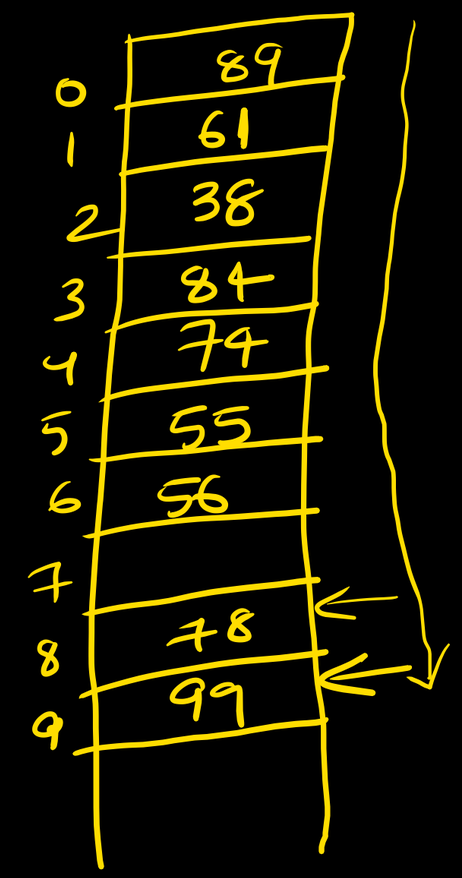
$= 9 + 4^2 = 5$

$= 9 + 6^2 = 5$

$= 9 + 7^2 = 8$

$= 9 + 8^2 = 3$

Hence even though there is space we cannot put the element.



Double Hashing:

HT[10] 0-9

Key = 55, 99, 61, 89, 78, 74, 38, 56, 84

$HF_1(key) = key \bmod m$ (m = 10)

$HF_2(key) = 1 + (key) \bmod (m-2)$ (m-2 = 8)

CRT = DH

$DH(key, i) = (HF_1(key) + i \cdot HF_2(key)) \bmod m$

DH(55, 0) = 5

DH(99, 0) = 9

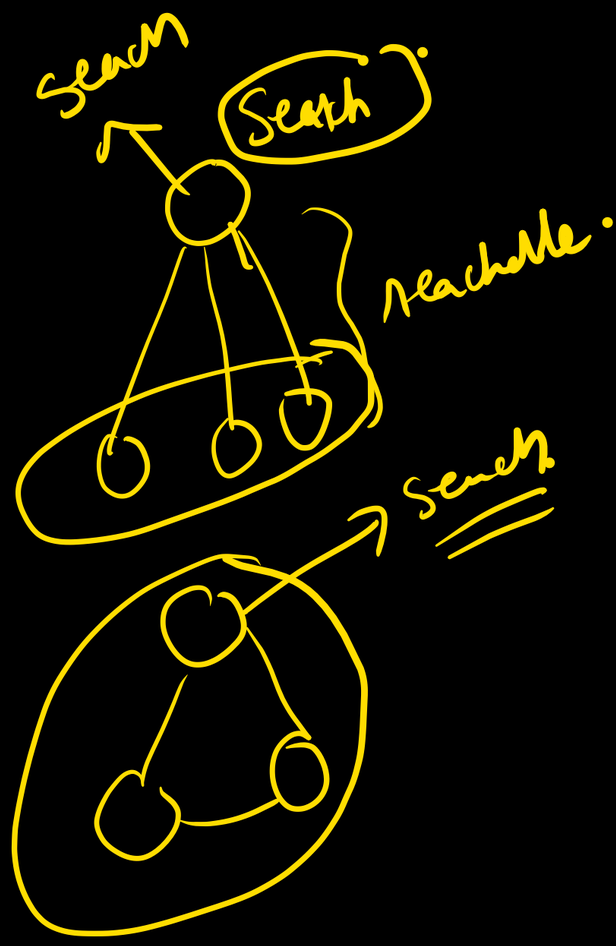
DH(61, 0) = 1

DH(89, 0) = 9

DH(89, 1) = 9 + 1.2 = (1) ✓

DH(89, 2) = (9 + 2.2) mod 10 = (3) ✓

0	
1	61
2	38
3	89
4	74
5	55
6	56
7	
8	78
9	9



BFS DFS

Other graph algo

{ Preorder traversal
Postorder traversal
Inorder traversal }

Breadth First Search
Depth First Search

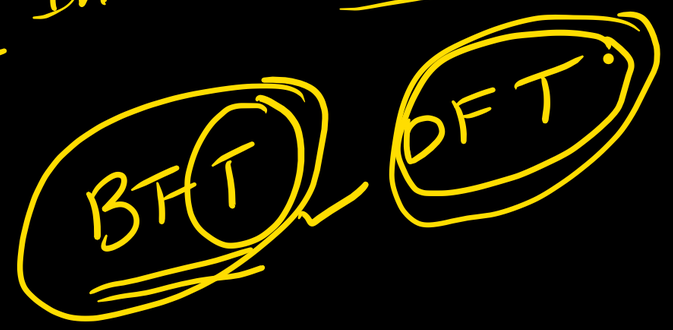
Tree is always connected.

↓

all nodes

↓

Trees.

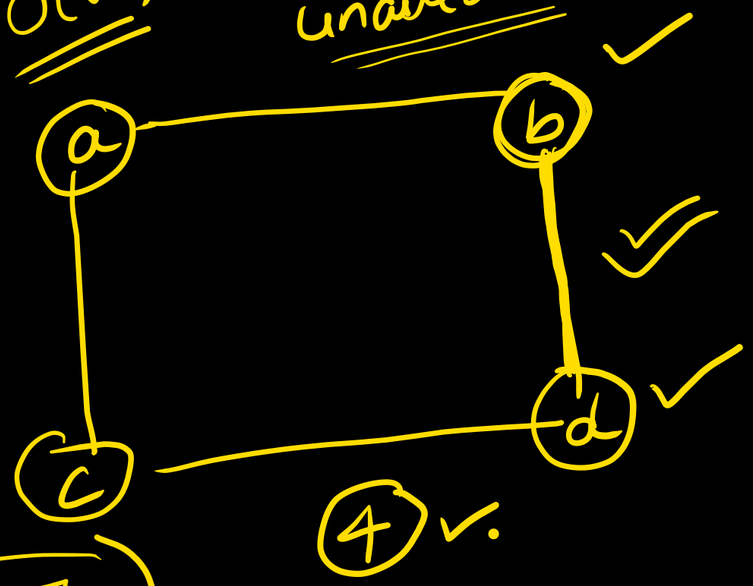


adjacency matrix

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

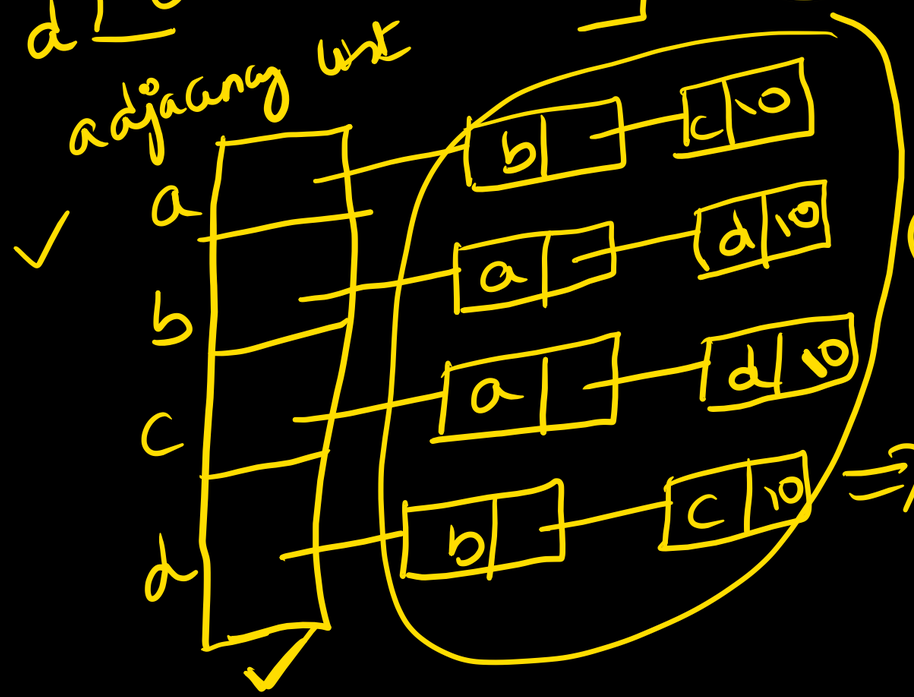
$\Rightarrow O(V^2)$

undirected



4 ✓

adjacency list



2E

$\Rightarrow O(V+2E)$

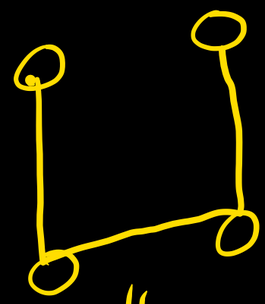
$= O(V+E)$

dense

$O(E) = O(V^2)$

$E = O(V)$

adjacency matrix.



adjacency list

BFS

Visited

↓

we have seen it

unexplored node

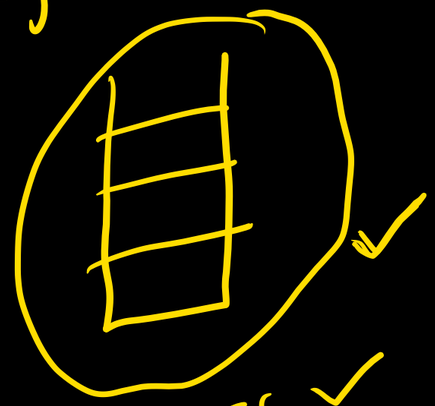
BFS ✓

DFS

Explored

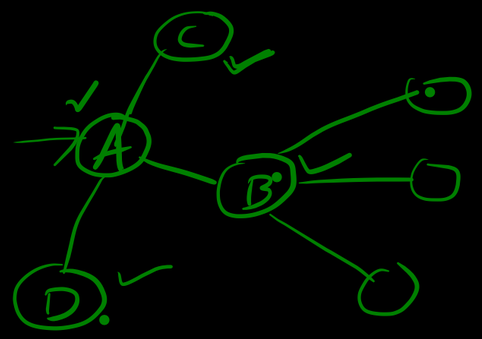
↓

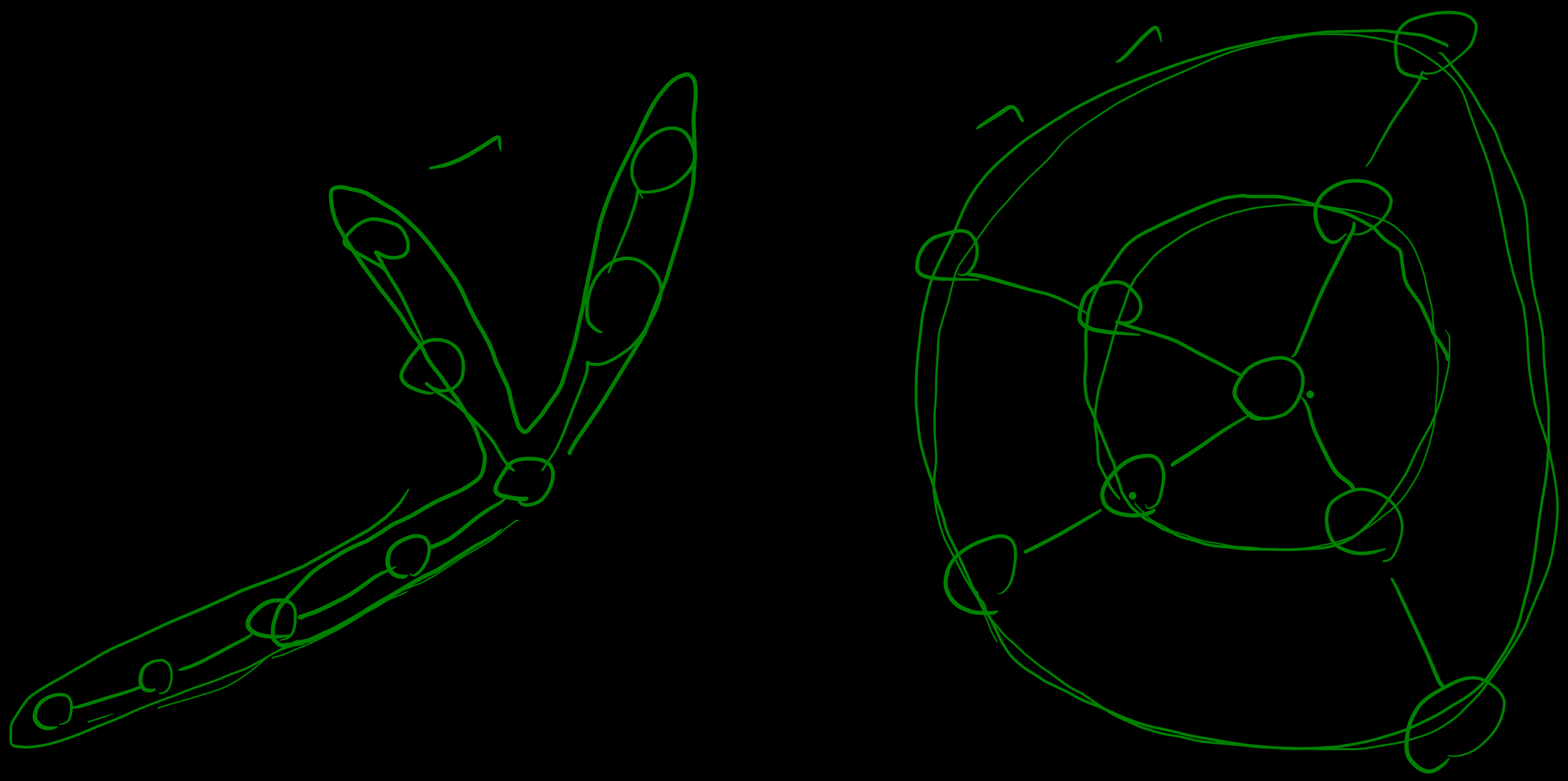
we have seens all the nodes adjacent to it



DFS ✓

```
BFS(v)
// 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 do
   {  if (visited[w] == 0)
   {  add w to q;
      visited[w] = 1;
   }
   if q is empty then return;
   Delete the next element, u, from q;
   }
}
```



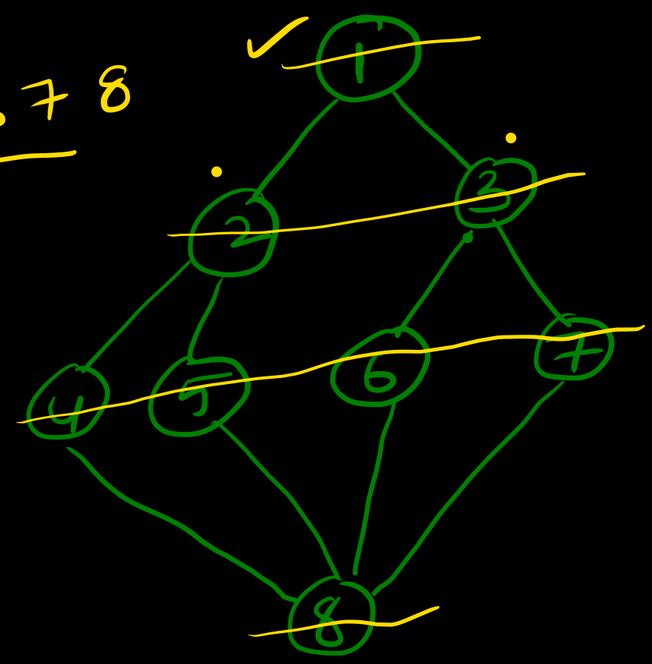


visited

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

$\Rightarrow O(V)$

1 2 3 4 5 6 7 8



2 - {1, 4, 5}
3 - {1, 6, 7}

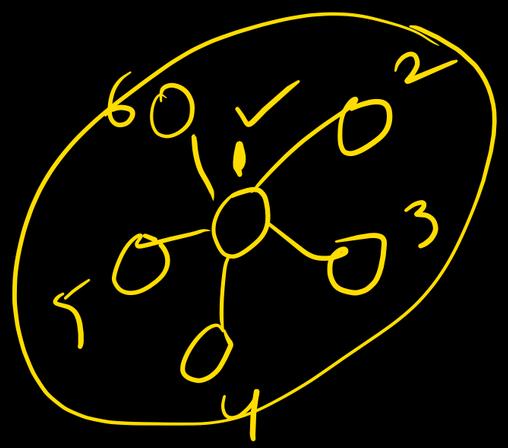
1	2	3	4	5	6	7	8
✓	✓	✓	✓	✓	✓	✓	✓

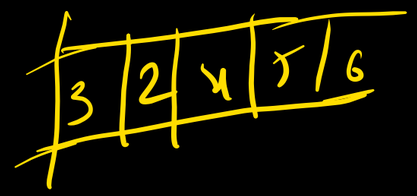
Queue:

4 - {2, 8}
5 - {2, 8}
6 - {3, 8}
7 - {3, 8}
8 - {4, 5, 6, 7}

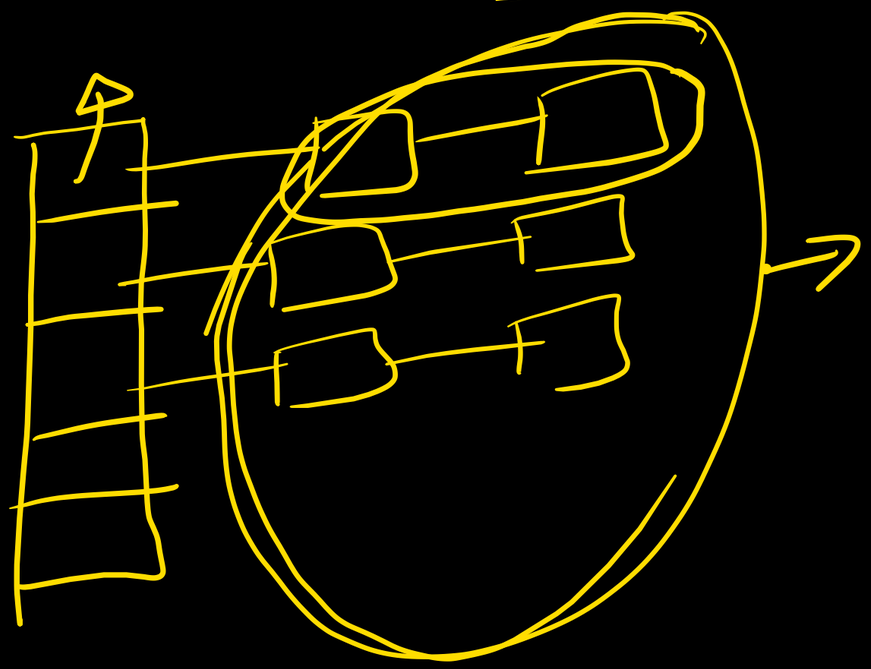
SC = Visited → O(n)

que → O(n).



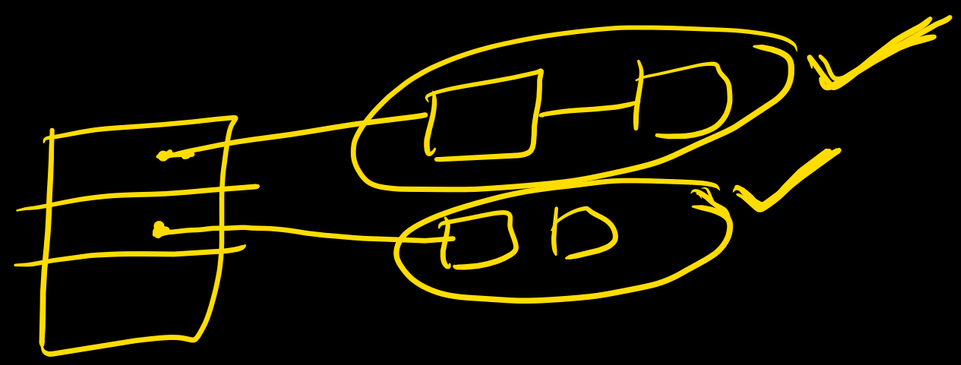


initialize visited → O(n) ✓



O(2E)
= O(E)

O(V + E)
↓ ↓
visited adj. edges

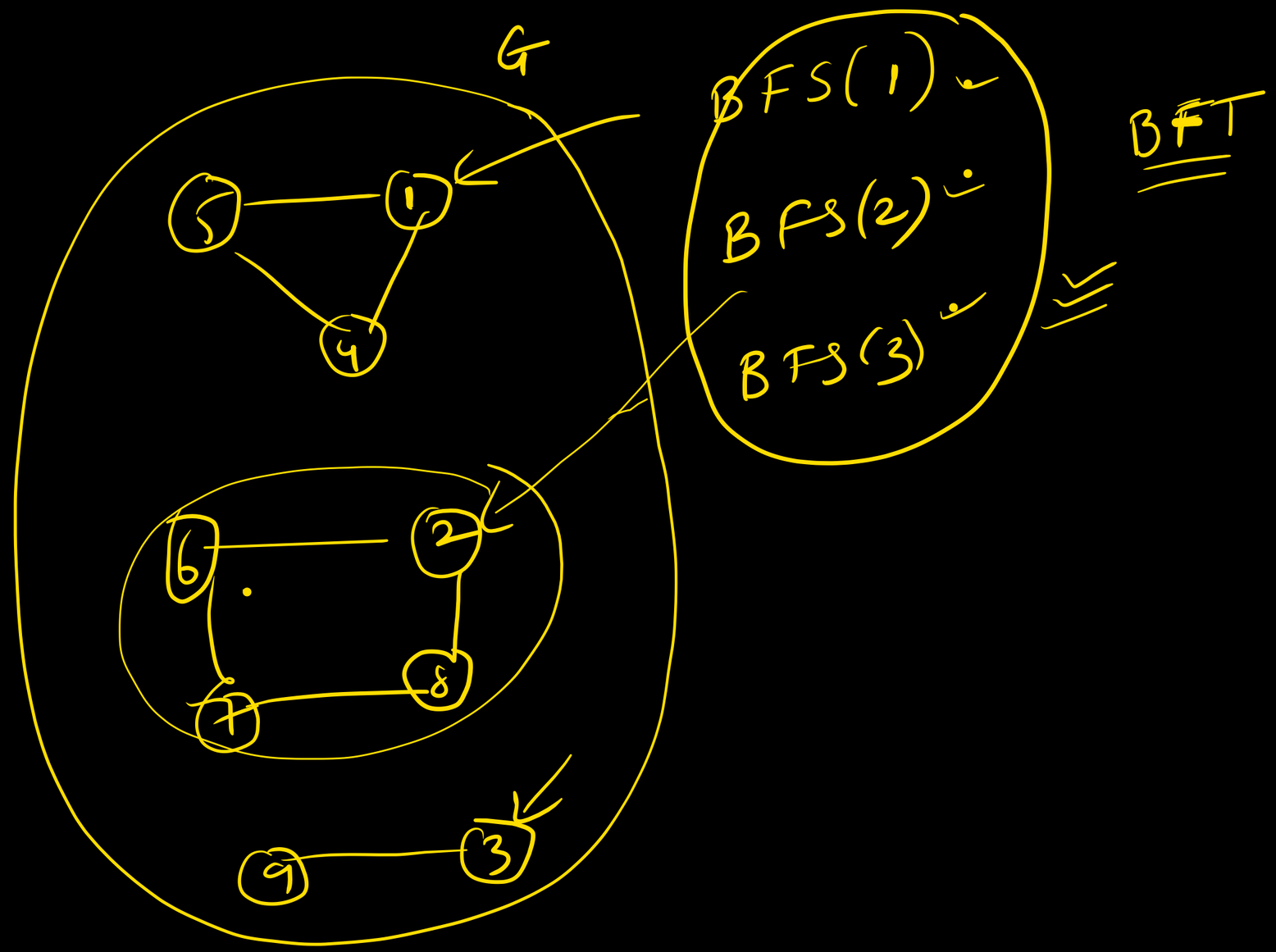


adj matrix $O(V^2)$

$O(V) \rightarrow$ unrelated

$$O(V^2 + V)$$

$$O(V^2)$$

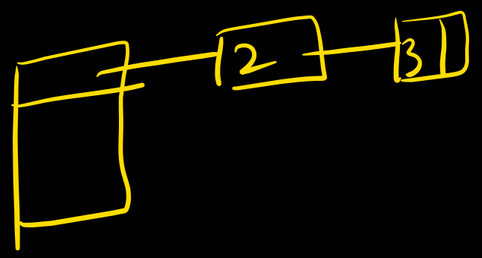

G
5
1
4
6
2
8
7
9
3
BFS(1)
BFS(2)
BFS(3)
BFT

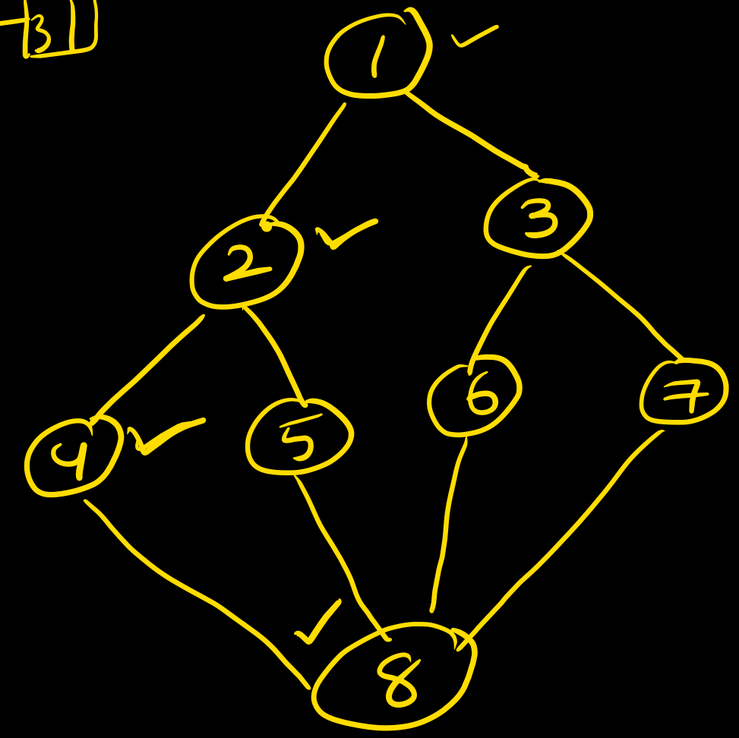
```
BFT(G,n)
{
  for(i=1 to n do)
    visited[i]=0;
  for(i=1 to n do)
    if(visited[i]==0) then BFS(i);
}
```

I) II) III) IV)

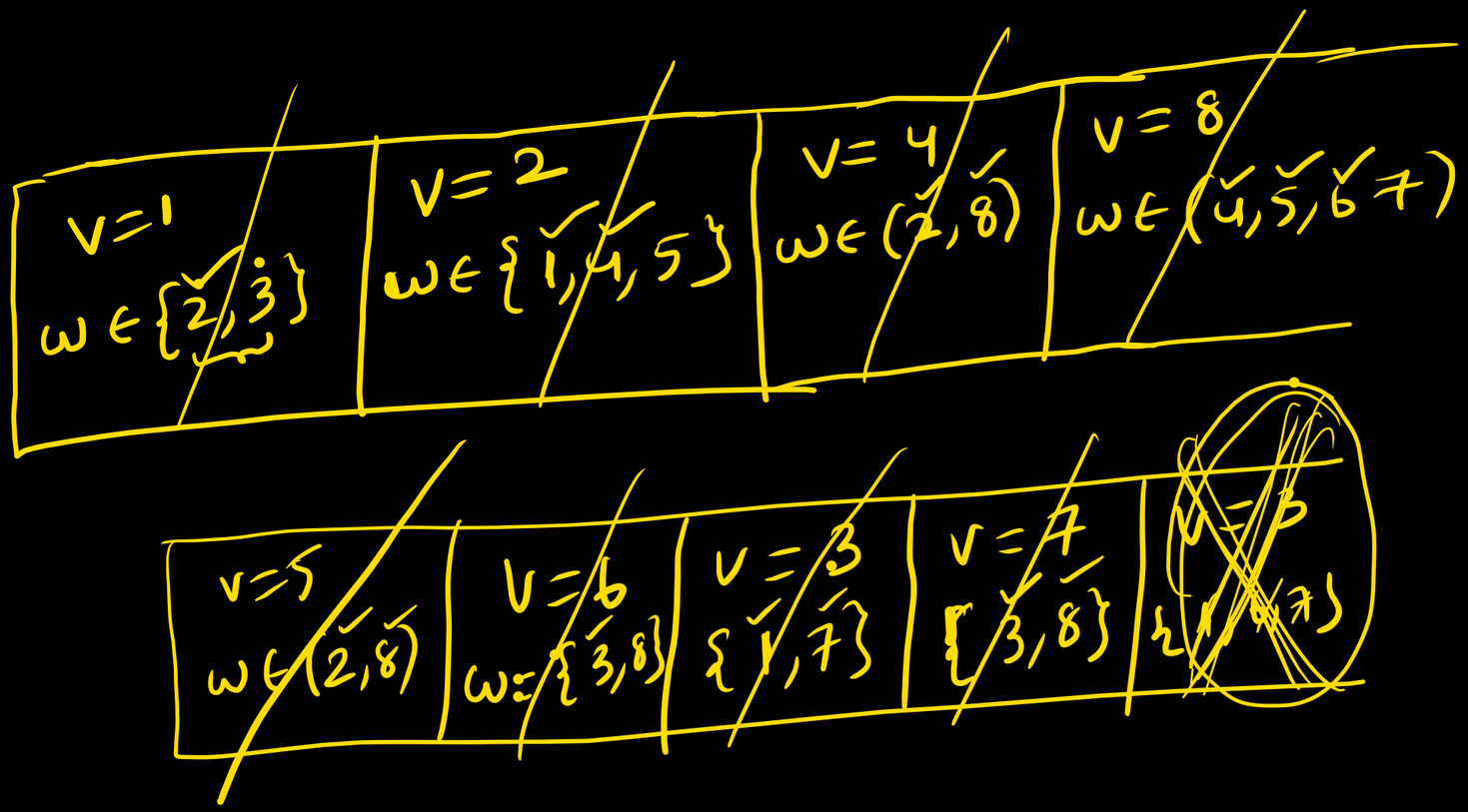
```
DFS(v)
{   visited[v] = 1 ;
    for each vertex w adj v do
     { if (visited[w] == 0) then
          DFS(w)
     }
}
```

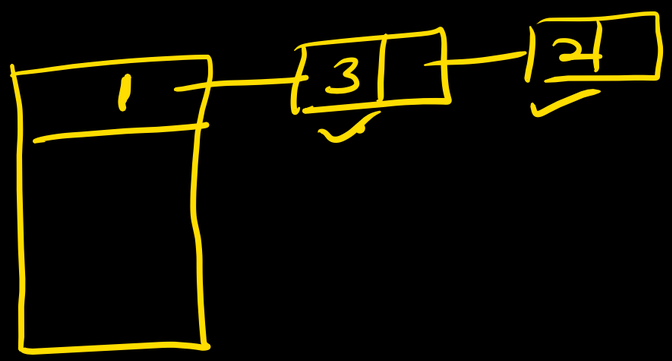

1 2 3 4 5 6 7 8

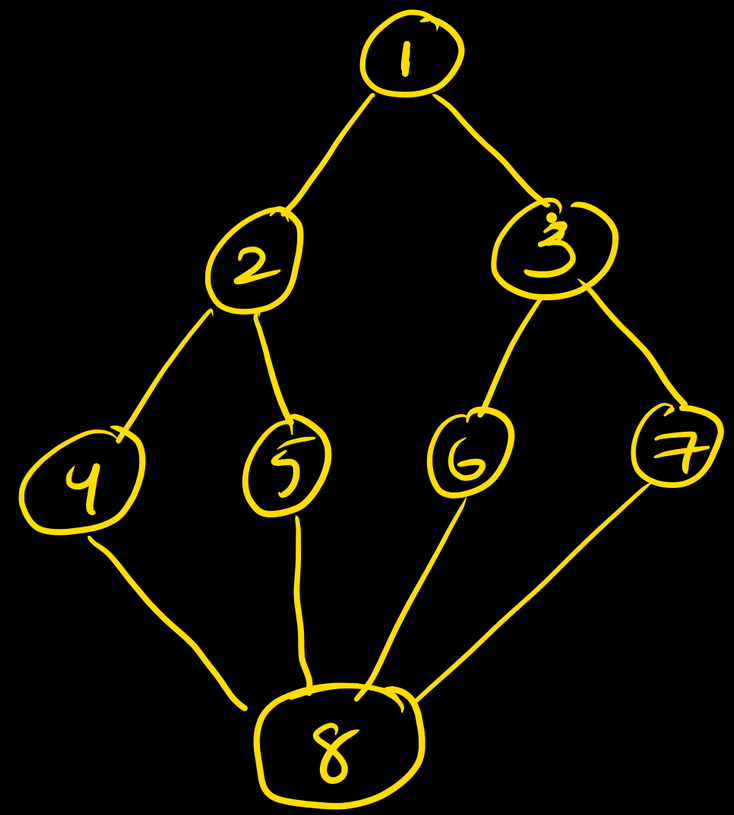

2
3


1
2
3
4
5
6
7
8

1 2 4 8 5 6 3 7

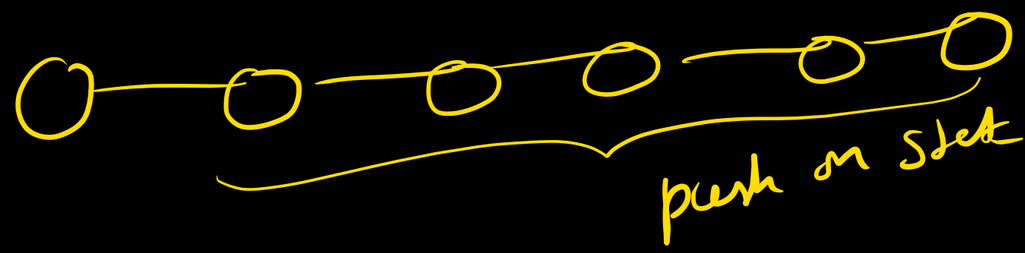

v=1
w∈{2,3}
v=2
w∈{1,4,5}
v=4
w∈(2,8)
v=8
w∈(4,5,6,7)
v=5
w∈(2,8)
v=6
w∈{3,8}
v=3
{1,7}
v=7
{3,8}


1
3
2


1
2
3
4
5
6
7
8

SC of DFS : $O(n)$ → visited ✓

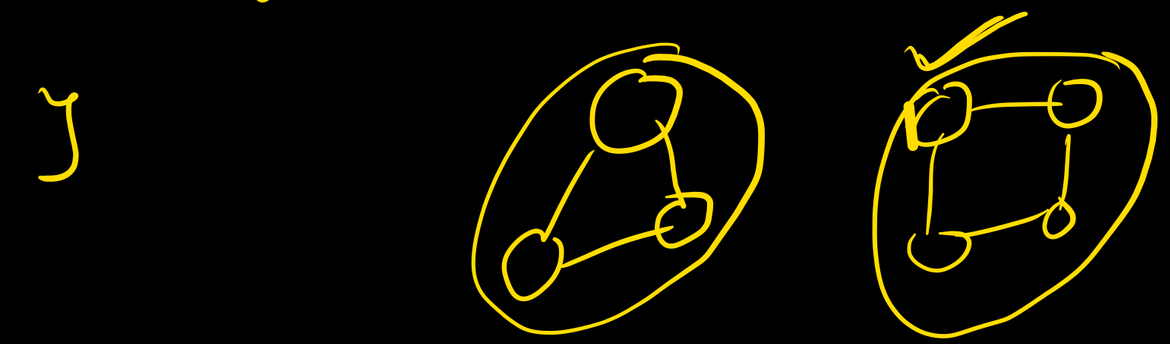
$O(n)$ → stack ✓



adj list :- $O(2E) + O(V) \Rightarrow O(E+V)$

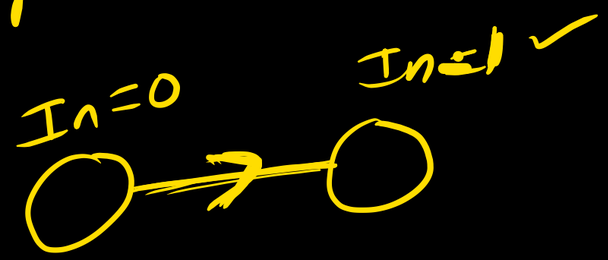
adj matrix :- $O(V^2) + O(V) = O(V^2+V) = O(V^2)$.

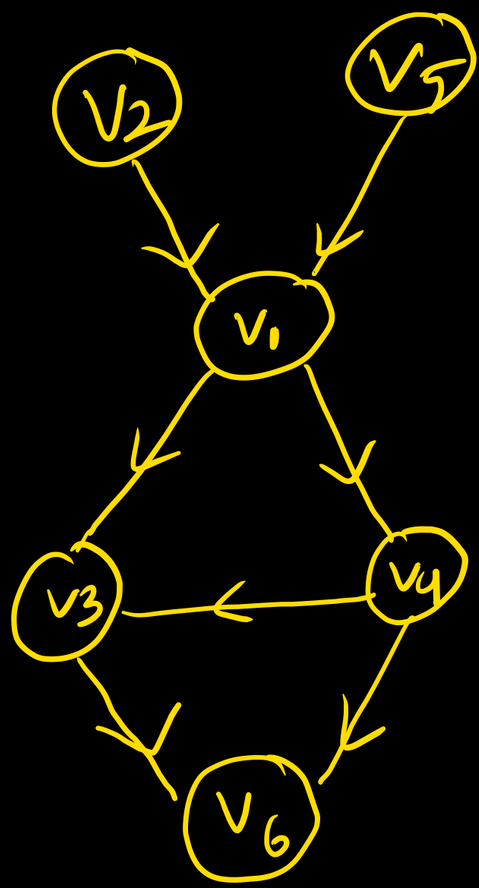
```
DFT(G,n)
{   for (i=1 to n do)
        visited[i]=0;
    for (i=1 to n do)
        if (visited[i]=0) then DFS(i);
}
```



Topological sort on DAG

Directed acyclic graph.





V_2 V_5 V_1 V_4 V_3 V_6

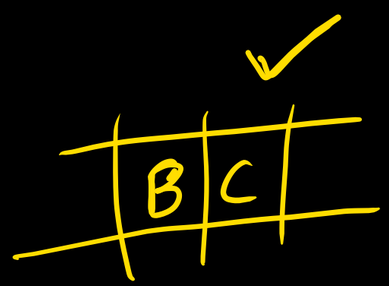
V_5 V_2

algo:

Step 1: Identify vertices then have no incoming edges — $O(V) \times V = O(V^2)$ (O(1))

✓Step 2: Delete this vertex of indegree 0 and all its outgoing edges from the graph. O(E) ✓

Repeat ① and ② until graph is empty



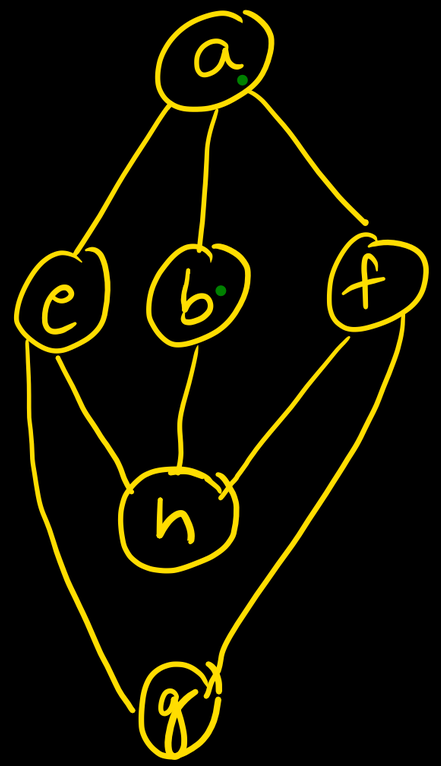
✓

queue.

$O((V^2 + E))$.

V

$O(V+E)$



I a b e g h f

a	b	h
e b f	a h	

II a b f e h g e

a	b
e b f	a h

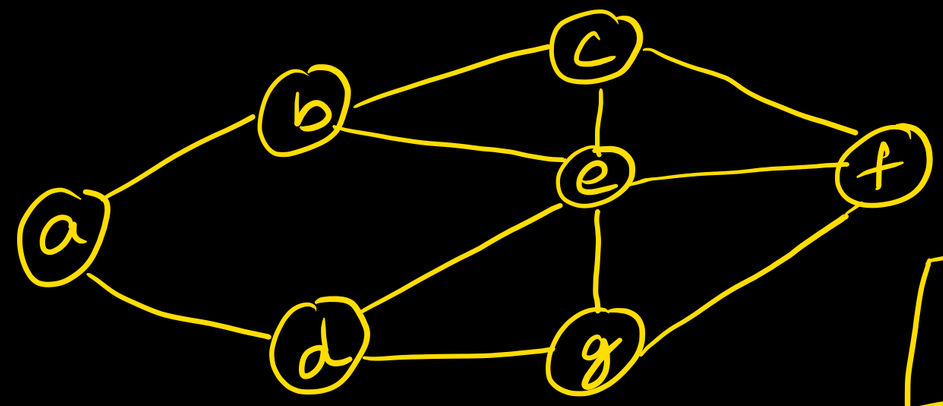
III a b f h g e

a	b
e b f	a h

IV a f g h b e x

a	f
e f b	a h

gdz:



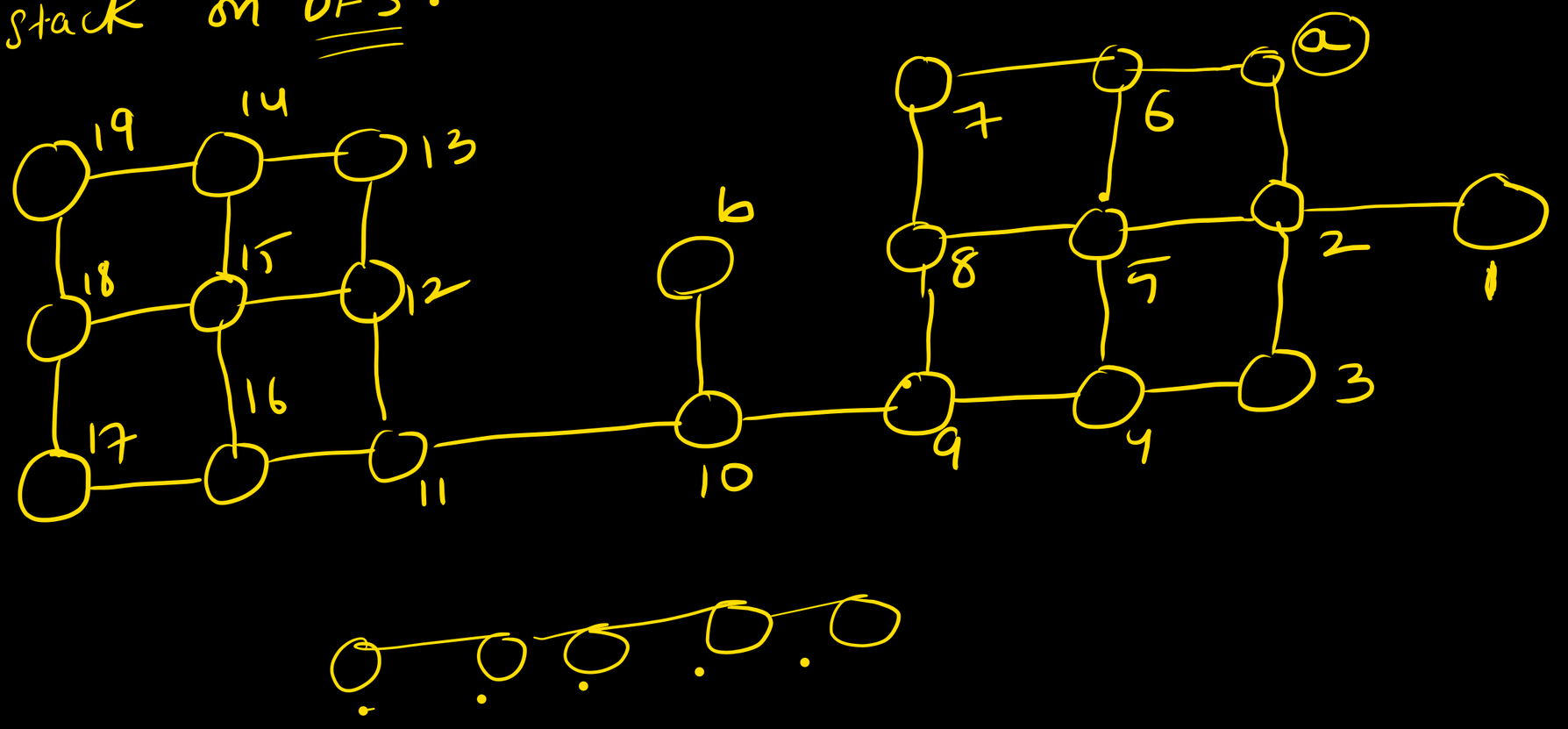
DFS? a b e f c g d

a	b	e	f	c	g
bd	ace	bcfdg	ceg	bef	def

1) a b e f d g c

a	b	e	f
{b, d}	{a, c, e}	b, d, g, c, f	c e g

what is the max no of nodes that can be present on recursion stack on DFS.



Q6 Consider a DFS of an undirected graph with 3 vertices, P, Q, R. Let the discovery time d(u) represent the time instance when the vertex 'u' is first visited and finish time f(u) represent the time instance when vertex 'u' is last visited:

Given that

d(P) = 5
d(Q) = 6
d(R) = 14

↓ Pushed

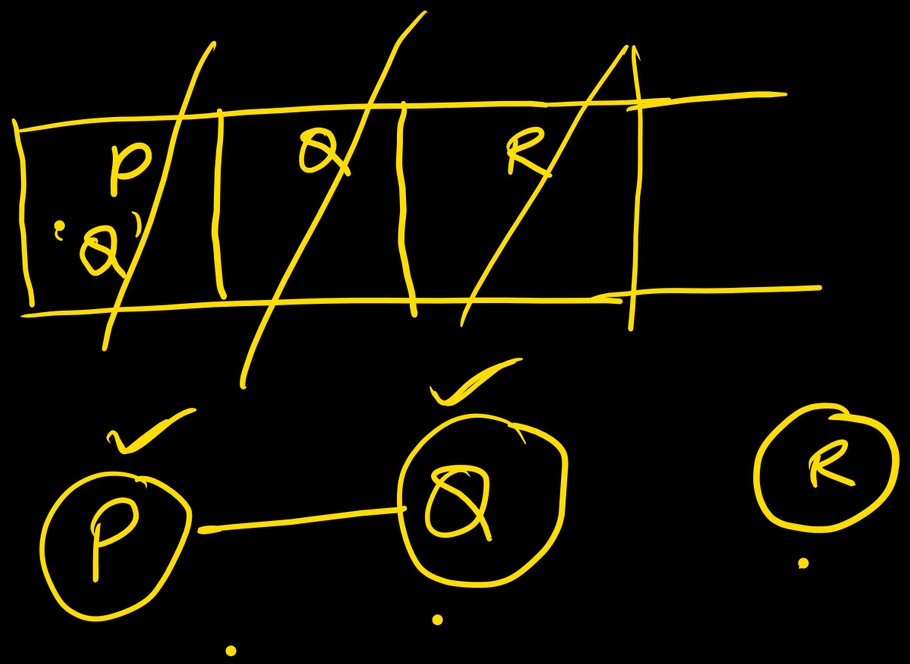
f(P) = 12
f(Q) = 10
f(R) = 18

↓ popped

what is the graph

5 6 10 12 14 18

d(P) d(Q) f(Q) f(P) d(R) f(R)



DFT