Assignment 3

SHUBHAM SHANKAR
ID : 1001761068
ASSG NO: 03

① Maze generation

Step 1 : Initialize a grid of square,
Step 2 : Start with entire grid subdivided into Squares

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

Step 3 : Represent Each Square as seperate disjoint set

{0} {1} {2} -- {19}

Step 4: Randomly choose a cell & mark it as a current cell

Step 5 : Initialize no of visited cell
Step 6 : Repeat the following.

Algorithm

① Start at a random cell & mark it as a current cell

② mark the current cell as Visited & get the list of its neighbours.

③ For each neighbour, Start with a

randomly selected neighbour.

ⓐ If Neighbour hasn't been Visited

① push the Neighbour into the ~~stack~~ Queue

② Remove the wall b/n this cell & the neighbour

③ Mark Neighbour as Visited

④ Make it the Current Cell

ⓑ Else

① pop the cell

② Make it as current cell

Pseudo Code

Initialize a random node

Mark it as a current Cell (C).

Mark the node as Visited (V)

For Every Neighbor (n) do:

if n ≠ Visited then,

push neighbour (n) into Queue (Q)

Remove the wall b/n current & neighbouring node

Mark Neighbours as Visited,

Mark Neighbour (n) as Current Cell (C)

Else do:

pop the cell & set the cell as current cell (C)

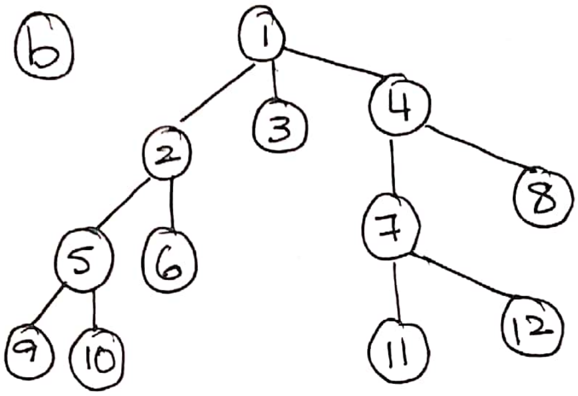
Scanned by CamScanner

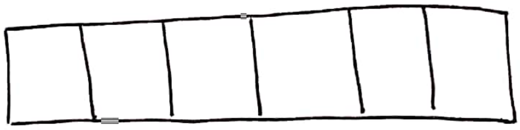
End if
End for
End

② ⓐ Pseudo Code

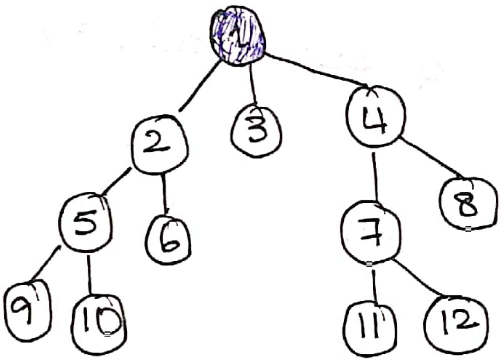
```
Mark all Vertices as Unvisited
for each V set dist (V) = ∞
Initialize Search Tree T to be empty.
Mark Vertex S as Visited & Set dist(S)=0
enq(S)
while Q is nonempty do:
    U = deq(Q)
    for each Vertex v ∈ Adj(U) do:
        if V is not visited do:
            add edge (U,V) to T
            Mark V as Visited, enq(V)
            set dist(V) = dist (U) + 1
        End if
    End for
End
```

(b)

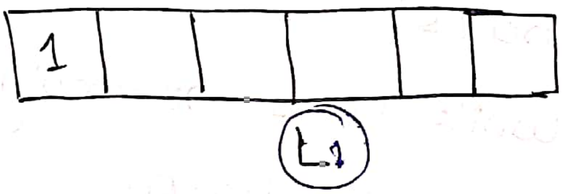




Step 1 : 1 form's the 0^{th} Layer L_0



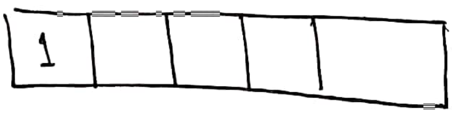
- push 1



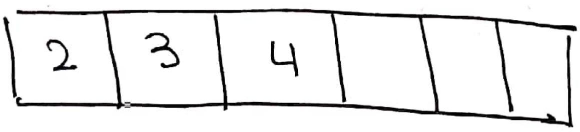
The Unvisited Vertex , 2, 3, 4 are adjacent 1.
These form first Layer L_1

- POP 1
- push 2, 3, 4

pop 1 ←



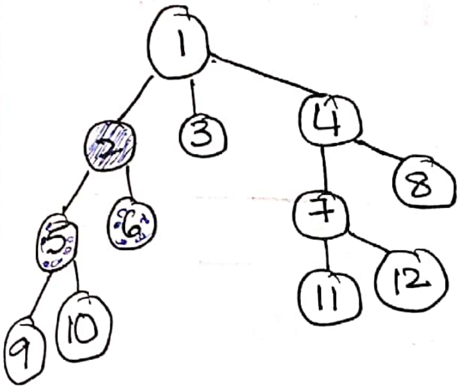
← Push 2, 3, 4



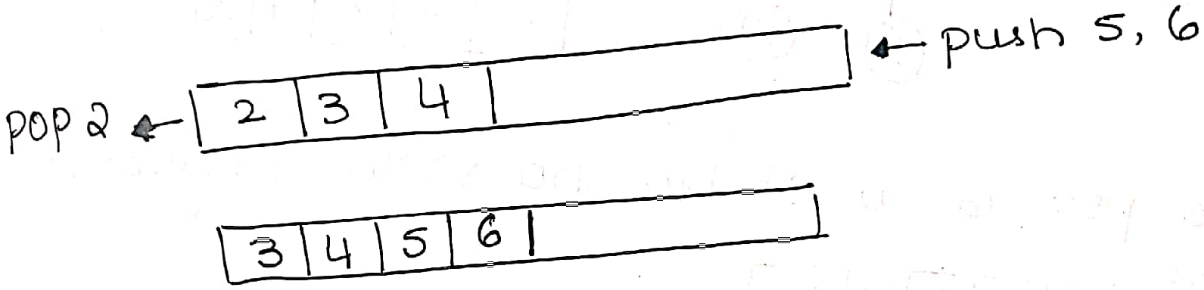
L_2

Scanned by CamScanner

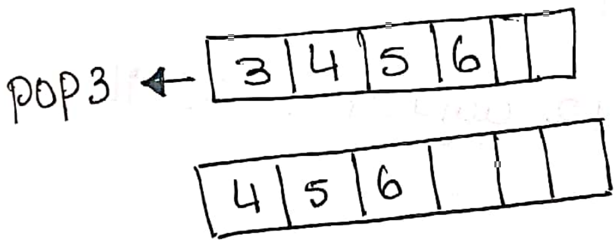
Step 2 : (a) We now begin popping L_1 Vertices.



- POP 2
- 5 & 6 is adjacent to 2, so it will be tagged Layer 2
- push 5, 6

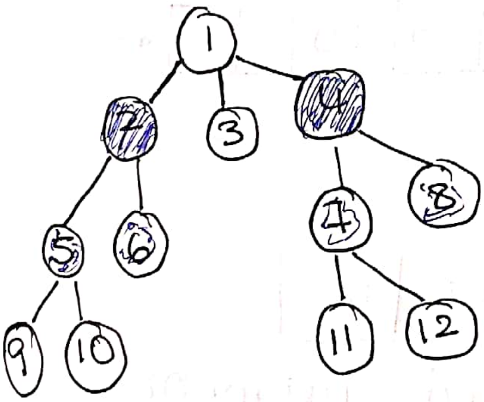


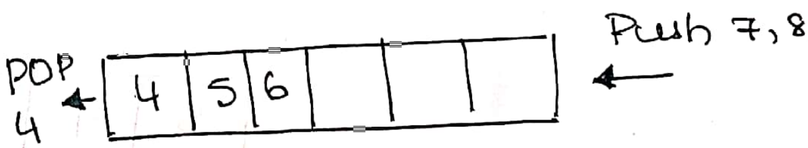
(b) We pop 3, which has no other Unvisite neighbour

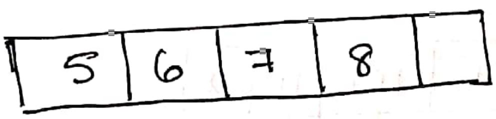


(c) poping 4 pusher 7 & 8 to the queue & tag it to Layer 2

- POP 4
- push 7, 8



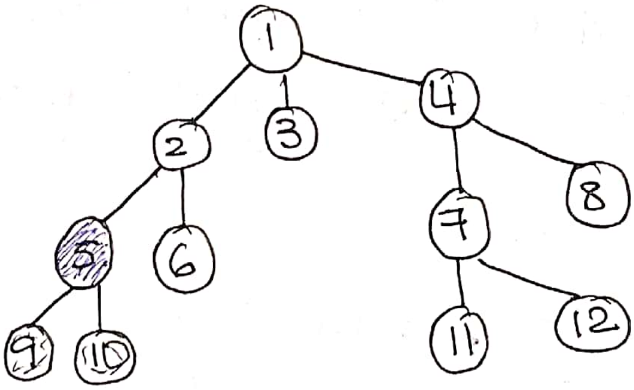




L3

Step 3 : (a) We now begin poping L_2.

pop 5, 9, 10 is adjacent to 5, so it is tagged to Layer 3



- POP 5

POPS ←

5	6	7	8	

← Push

Push 9, 10

6	7	8	9	10

(b) We pop 6 as it has no visited neighbour

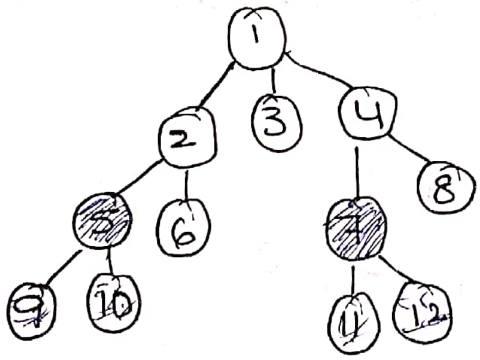
POP 6 ←

6	7	8	9	10

7	8	9	10	

(c) POP 7, it has 11 & 12 adjacent, so it will join Level 3

- POP 7,
- Push 11, 12



POP 7 ←

7	8	9	10		

← Push 11, 12

8	9	10	11	12

(d) POP 8, as it has no visited neighbor

POP 8 ←

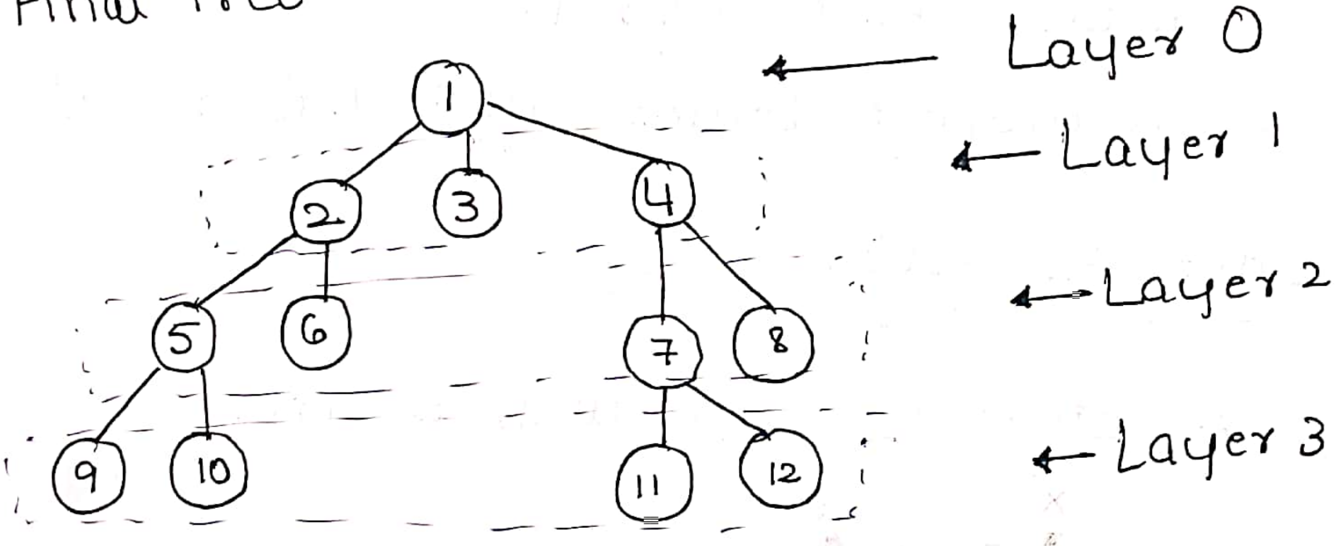
8	9	10	11	12

9	10	11	12	

Thus the final Layer 3 contain

9	10	11	12

Final tree



Distance from any Layer 1 Vertices to Vertex 1 is $\boxed{1}$

Distance from any Layer 2 Vertices (ie, 5,6,7,8) to Vertex 1 is $\boxed{2}$

Distance from any Layer 3 Vertices (i,e 9,10,11,12) to Vertex 1 is $\boxed{3}$

③ Cyclic Graph means a graph that Contains a cycle i.e, some no of Vertices is Connected in a close chain.

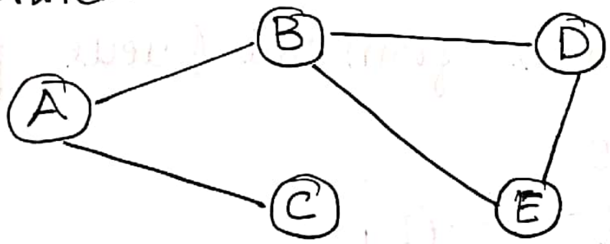
→ 2 approach used to figure out if a graph contain cycle or not is using

① Breadth first Search [BFS]

② Depth first Search [DFS]

① BFS

Consider a Undirected Graph



• We use queue here.

• −1 = Unvisited

0 = Visited & in queue

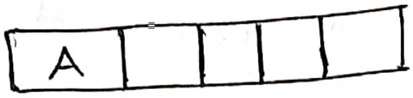
1 = poped out of queue.

→ Step 1 : Set all the node to −1

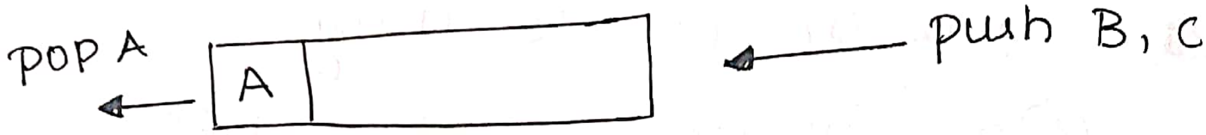
i.e Consider all the node to Unvisited initially

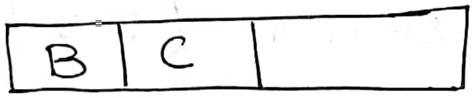
→ Step 2 : Start from Node A.

push A into the queue.



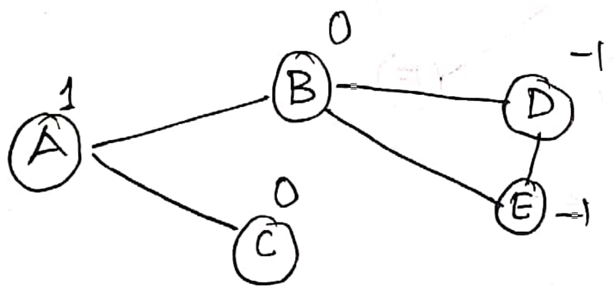
→ **Step 3** : push the neighbouring Unvisited Vertex of A into Queue. & Pop A.



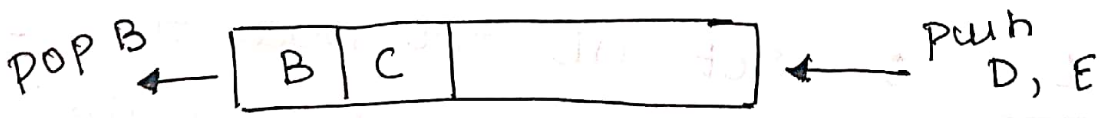


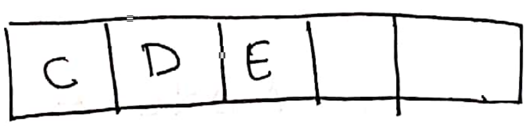
When we push Vertex in Queue, flag changes to Zero

When we pop Vertex from the Queue flag changes to 1

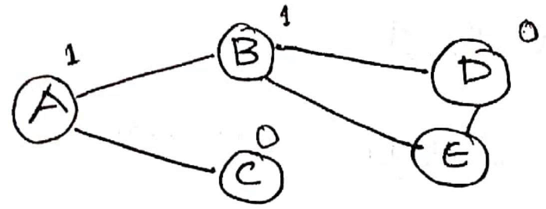


Step 4 : push the neighbouring Unvisited Vertex of B into the Queue & pop B





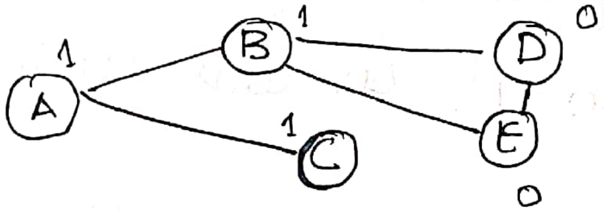
- Flag of B becomes 1 & flag of D, E = 0



Step 5 : pop C, & it doesnot have any Unvisited Vertex

C	D	E	

→ pop C



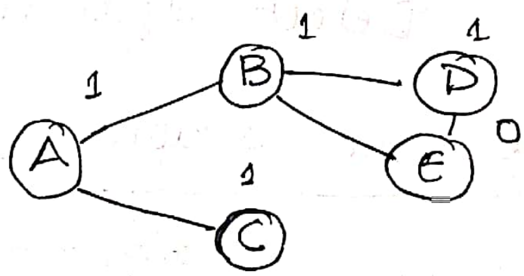
- Flag of C Changes to 1

D	E	

Step 6 pop D, & it does not have any Unvisite Vertex, But have a Visited Vertex

POP D ←

D	E	



E	

Flag of D change to 1

Step 7 : Here, we find that the adjacent Node of D is E and has flag 0.

i.e E ~~means~~ has flag 0 means, it is already present in the queue

& node Can only Enter a queue, when

it is adjacent to a node.

→ So there this will show that the
∴ graph has a cycle

The Condition :

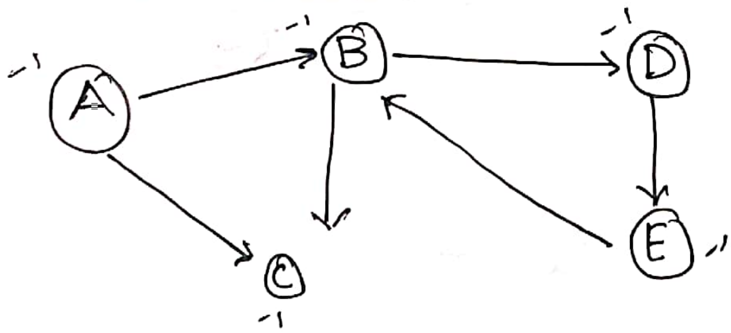
If any Vertex finds its adjacent Vertex with flag 0, then it contains cycle.

The above graph satisfies the equation
∴ have a cycle.

(b) DFS

Detect cycle in Direct graph.

Consider Graph



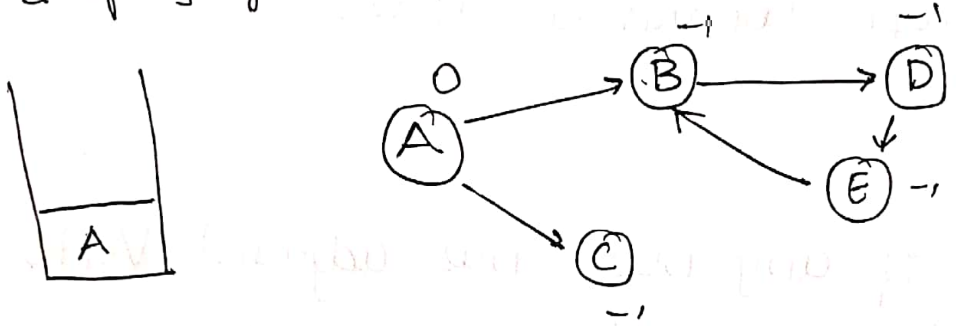
→ We use Stack

→ -1 = unvisited

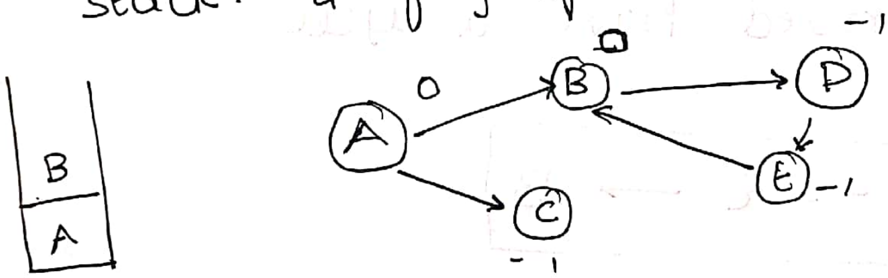
0 = Visited

1 = poped out

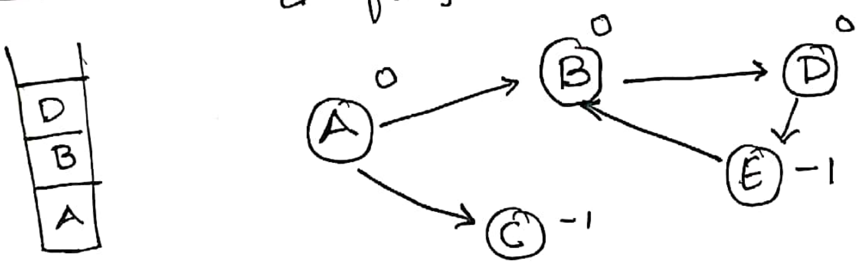
Step 1 Start with node A,
set all the node to -1, ie Unvisited
push node A to the stack.
& flag of A becomes 0.



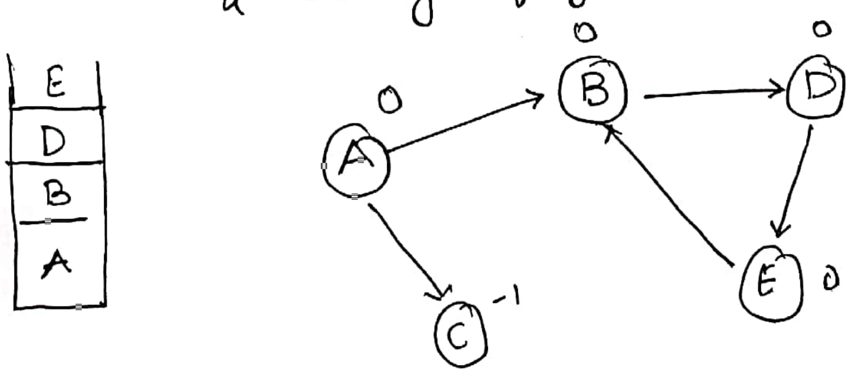
Step 2 : push the adjacent Vertex of B into stack. & flag of B becomes 0



Step 3 push the adjacent Vertex of B into stack & flag D becomes 0



Step 4 push the adjacent vertex of D into stack & change flag to 0



(Step 5 : Now, we see that E has a adjacent Vertex B, But B has a flag 0, which means it has Been Visited,

Hence the Graph Contain a cycle.

Condition

In DFS. if any node has adjacent Vertex with flag zero, then it has a cycle.

∴ The above directed have a cycle

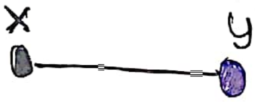
$$\boxed{B \rightarrow D \rightarrow E \rightarrow B}$$

④ Yes, any tree with 2 Vertices is a bipartite graph.

- A tree is a Connected graph with No cycle
- There is a Unique path b/n any 2 Vertices
- Every Tree is bipartite
- A graph is bipartite iff it has no odd cycle

Consider the following Eg :

① A bipartite graph with 2 Vertices



$V_1 = (x)$ $V_2 = (y)$

- Set formed by Vertices of a tree V_1 & V_2 are Mutually exclusive & Mutually exhaustive

i.e V_1 intersection V_2 = null

and $V_1 \cup V_2$ = Sample space

∴ any tree with 2 Vertices is a bipartite graph

This is alway true Even when the no of Vertices increase.

Scanned by CamScanner