## BFS using queue:-

a) a boolean array visited which will be true, if the node is visited false otherwise

b) Initially queue is empty

front = -1 , rear = -1

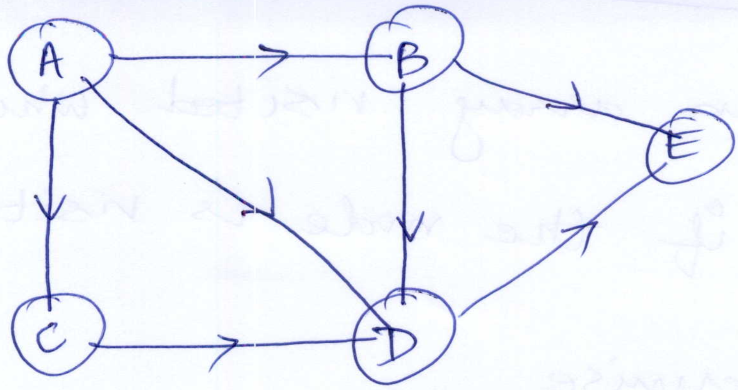
c) visited [i] = false, for all i = 1 to n,

n is total no. of nodes.

**Procedure:-**

1. Insert starting node into the queue.
2. Delete front element from the queue
   Insert all its unvisited neighbours into the queue at the end, and traverse them. Make visited true for these nodes.
3. Repeat step 2 until the queue is empty.

Ex:



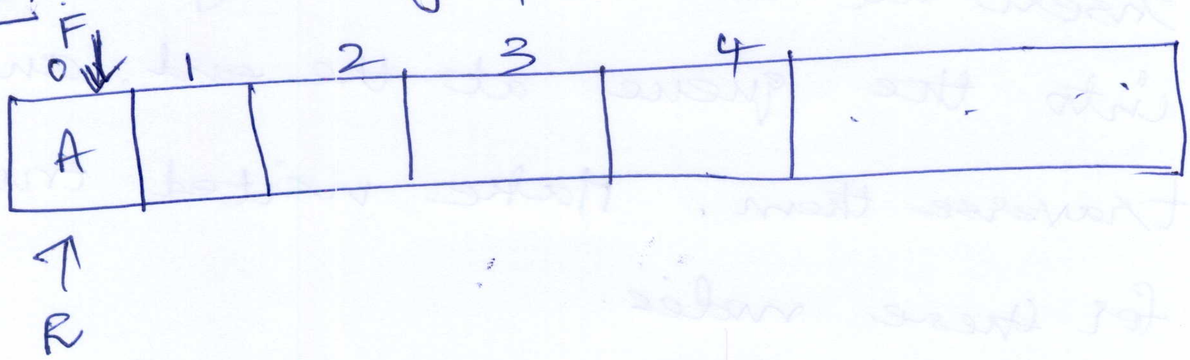

Adjacency list of the graph:-

| Vertex | Adjacency List |
|---|---|
| A | B, C, D |
| B | D, E |
| C | D |
| D | E |
| E | — |

If source vertex is A,

Initially push A to the queue.

Step 1:

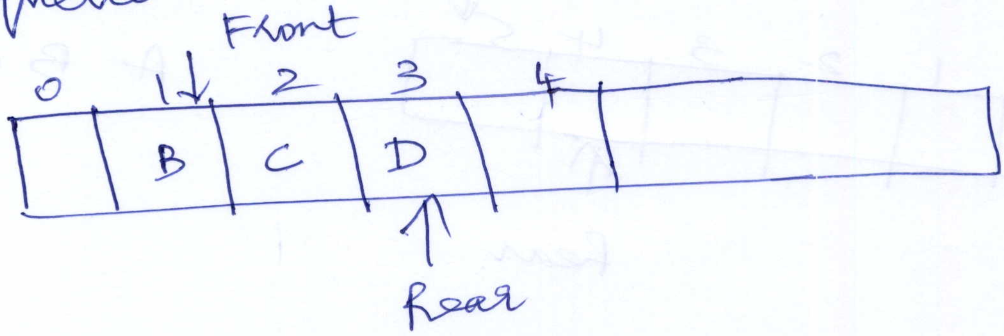



F=0

R=0

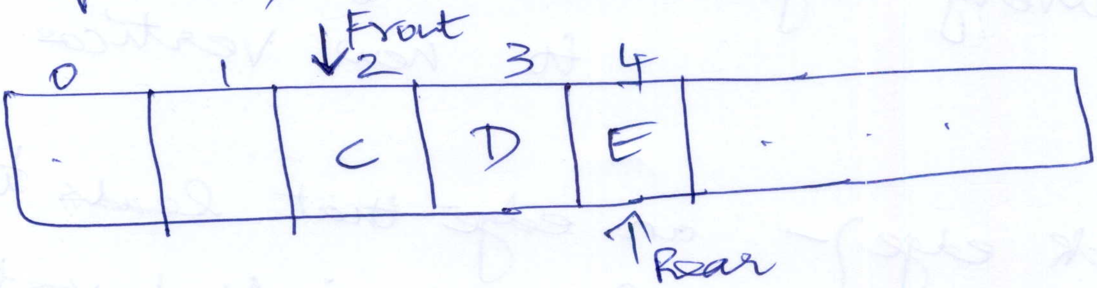
**Step 2:** Remove the front element A from the queue ( Front = Front + 1) and display it.

Push all neighbouring vertices of A to the queue ( Rear = Rear + 1), if it is not in queue.



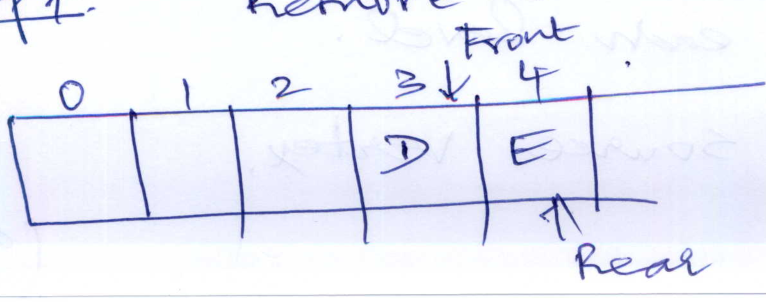

Traversal O/P = A.

**Step 3:** Remove B. Add neighbors of B to the queue, if not already




O/P: A B

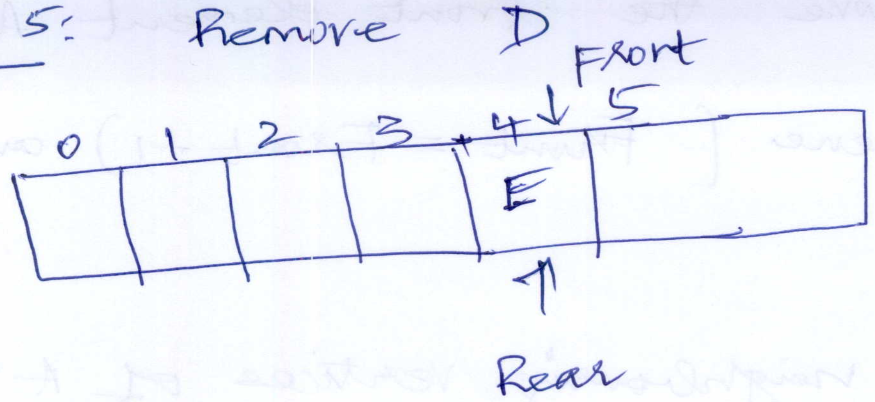
**Step 4:** Remove C




O/P:

A B C

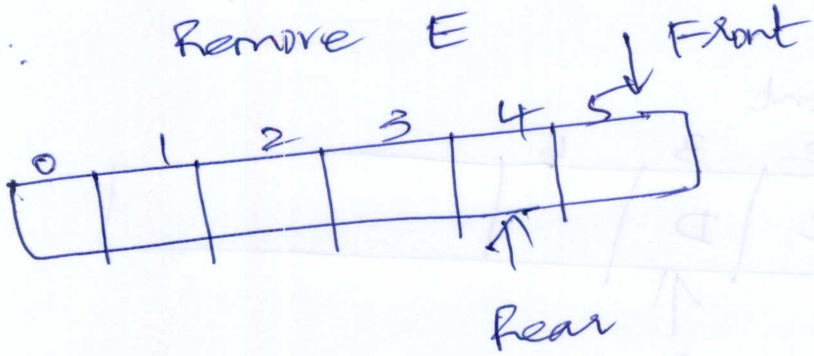
Step 5: Remove D




O/P:

A B C D

Step 6: Remove E




O/P:

A B C D E

Repeat until (Front > rear)

## Pseudo Procedure for BFS:-

Discovery edge – an edge that leads to new vertices.

back edge }
Cross edge } – an edge that leads to already visited vertices.

$L_0, L_1, L_2, \ldots$ – Containers of nodes in each level.

s – Source Vertex.

Algorithm BFS(G, s):

Input: A graph G and a vertex s of G.

Output: A labeling of the edges in the connected component of s as discovery edges and cross edges.

Create an empty container $L_0$
insert s into $L_0$
$i \leftarrow 0$.
while $L_i$ is not empty do
    Create an empty container $L_{i+1}$
    for each vertex v in $L_i$ do
        for all edges e in G.incidentEdges(v) do
            if edge e is unexplored then
                let w be the other endpoint of e
                if vertex w is unexplored then
                    label e as a discovery edge

insert w into $L_{i+1}$

else

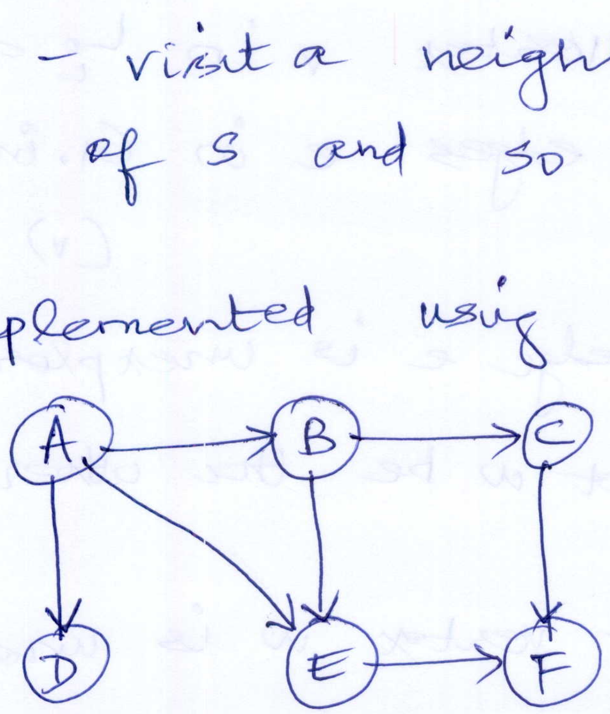
label e as a cross edge

$i \leftarrow i+1$.

**Depth - First Search**:- DFS - Recursive BackTracking

- Search deeper in the graph, whenever possible.
- If s is source vertex,
  - visit s
  - visit a neighbor vertex of s
    - visit a neighbor of a neighbor of s and so on.
- implemented using stack.

EX:




O/P:

A B E F C D

Using stack :-

array STACK :- keeps the unvisited neighbors of the node.

boolean array VISITED :- TRUE(node) if visited

false, otherwise

Initially stack is empty and TOP = -1.

Initially visited [i] = false for all i = 1 to n

Procedure :-

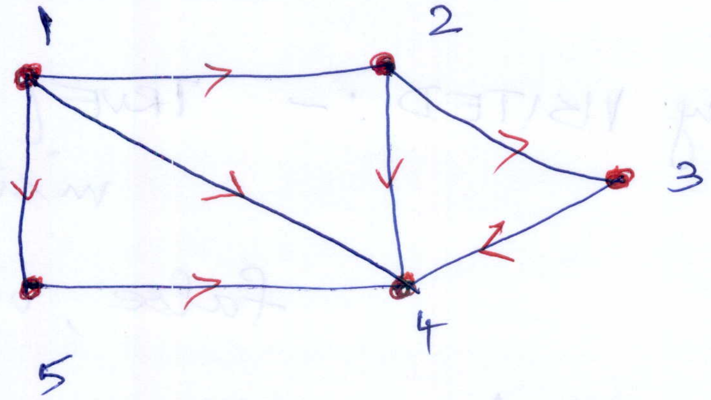
1. Push starting node into the stack
2. Pop an element from stack.
   If it is not traversed, traverse it, make visited for this node true.

   If traversed, ignore it.

3. Push all the unvisited adjacent nodes of the popped element. Push them even if already on the stack.

4. Repeat steps 3 and 4 until stack is empty.

Ex: Starting node is 1.



Step 1: Push node 1

| 0 | 1 | 2 | |
|---|---|---|---|
| 1 | | | ... |

Top

Top = 0

O/P: 1

STACK = 1.

STEP 2: • Pop node 1, ignore it

• VISITED[1] = True.

• Push all unvisited adjacent nodes of 1, namely 2, 5, 4.

| 0 | 1 | 2 | |
|---|---|---|---|
| 2 | 5 | 4 | ... |

Top.

STACK = 2, 5, 4

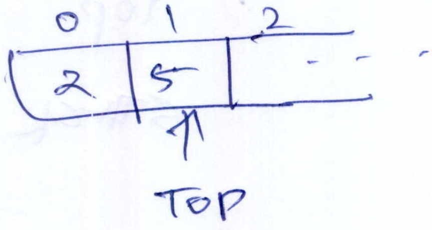
O/P: 1

STEP 3: Pop 4, Traverse, Push all its unvisited adjacent nodes.

visited [4] = True    O/P: 1, 4

Top = 1

Stack = 2, 5



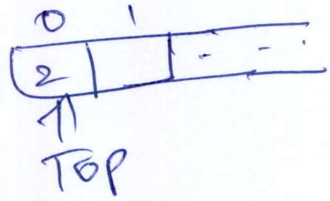

~~5~~

STEP 4: Pop 5, Traverse, {Push all its unvisited adjacent nodes

visited [5] = True

TOP = 0

Stack = 2




O|P: 1, 4, 5.

5 has 1 adjacent node 4, but its already visited

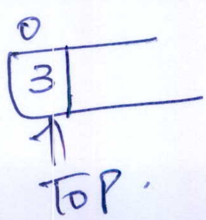
STEP 5: Pop 2, Traverse, Push all its unvisited adjacent nodes (3)

O|P: 1, 4, 5, 2

visited [2] = True

Top = 0

STACK = 3



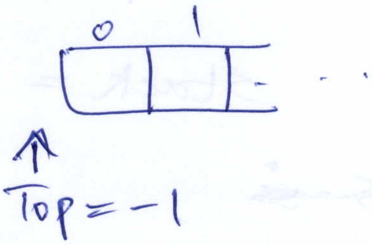
STEP 6: Pop 3, Traverse, push all its unvisited adjacent nodes (4, but its visited)

visited [3] = True

O/P: 1, 4, 5, 2, 3

∴ Top = -1

STACK = NULL




∴ Repeat until stack becomes empty.

~~Breadth~~ Depth - First Search: (DFS) - Recursive BackTracking

Algorithm DFS(G)

Input: graph G

Output: labeling of the edges of G as discovery edges and back edges.

for all u ∈ G.vertices()

setLabel(u, UNEXPLORED)

for all e ∈ G.edges()

setLabel(e, UNEXPLORED)

for all v ∈ G.vertices()

if getLabel(v) = UNEXPLORED

DFS(G, v)

Algorithm DFS (G, v)

Input: graph G and a start vertex v of G

Output: labeling of the edges of G in the connected component of v as discovery edges and back edges

setLabel (v, VISITED)

for all e ∈ G.incidentEdges(v)

if getLabel(e) = UNEXPLORED

setLabel(e, DISCOVERY)

DFS (G, w)

else

setLabel(e, BACK).