Graph operation

1. Insertion → add_node / add_vertex
   → add_edge
2. Deletion
3. Traversal

---

1. Function → Add a node (Adjacency Matrix representation)

```
nodes = [ ]      # store all nodes (A, B, C, D, E)
graph = [ ]      # store the matrix
node_count = 0
```

```
def add_node(v):
    global node_count
    if v in nodes:    # whether "v node" is present in list of nodes.
        Print(v, "is already Present in the graph")
    else:
        node_count = node_count + 1
        nodes.append(v)
        for n in graph:          # appending 0 in last column of matrix
            n.append(0)          # we need to add extra row and column to matrix (In the nested list append a zero to every inner list)
        temp = [ ]               # list to add row in matrix
        for i in range(node_count):
            temp.append(0)
        graph.append(temp)
```

```
def Print_graph():                    # To Print the Matrix
    for i in range(node_count):
        for j in range(node_count):
            Print(graph[i][j], end = " ")
        Print()

nodes = []
graph = []
node_count = 0
Print("Before adding nodes")
Print(nodes)
Print(graph)
add_node("A")
add_node("B")
Print("After adding nodes")
Print("nodes :", nodes)
Print(graph)
Print_graph()
```

O/P:

```
Before adding nodes
[]
[]
After adding nodes
nodes: ['A', 'B']
[[0,0], [0,0]]
0 0
0 0
```

(2) Function - Add a edge (Adjacency Matrix)

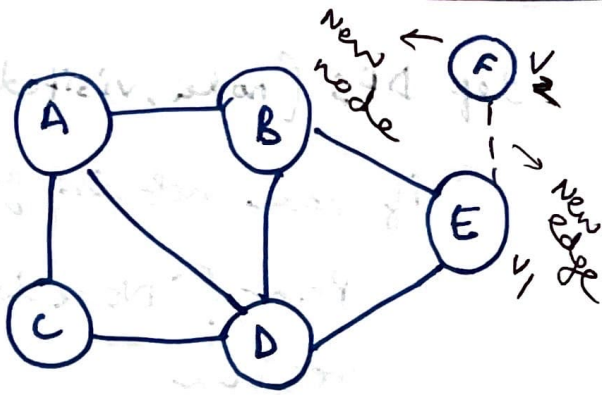
```
def add_edge (v1, v2):
    if v1 not in nodes:
        Print( v1, " is not present in graph " )
    elif v2 not in nodes:
        Print (v2, " is not present in graph")
    else:
        index1 = nodes.index(v1)
        index2 = nodes.index(v2)
        graph [index1] [index2] = 1
        graph [index2] [Index1] = 1

add_edge ("A", "B")
```

DFS using Recursion

→ Adjacency List Representation

→ Recursion



3

```
def add_node(v):
    if v in graph:
        Print(v, "is already present in graph")
    else:
        graph[v] = [ ]
```

4

```
def add_edge(v1, v2):
    if v1 not in graph:
        Print(v1, "is not present in graph")
    elif v2 not in graph:
        Print(v2, "is not present in graph")
    else:
        graph[v1].append(v2)
        graph[v2].append(v1)
```

```
graph = { A : [B, C, D],
          B : [A, E, D],
          C : [A, D],
          D : [A, B, C, E]
          E : [B, D] }  → ["B", "D"]
          F : [ ] → New node ["E"]
```

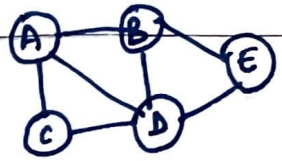
"F"

[illegible]

→ DFS Algorithm

Adding node to graph : Adding new row and column.

Deleting node to graph : In Adjacency matrix representation (delete a row and column in matrix)



(delete B)

Deletion node (Adjacency Matrix)

```
def delete_node(v):
    if v not in nodes:
        Print(v, "is not present in graph")
    else:
        index1=nodes.index(v)  # index of node, which has to be deleted
        node_count = node_count -1
        nodes.remove(v)
        graph.pop(Index1)  # Row is deleted
        for i in graph:
            i.pop(Index1)  # column is deleted
```

Deleting edge (Adjacency Matrix)

```
def delete_edge(v1, v2):
    if v1 not in nodes:
        Print(v1, "is not Present in graph")
    elif v2 not in nodes:
        Print(v2, "is not Present in graph")
    else:
        index1 = nodes.index(v1)
        index2 = nodes.index(v2)
        graph[index1][index2]=0
        graph[index2][index1] =0
```