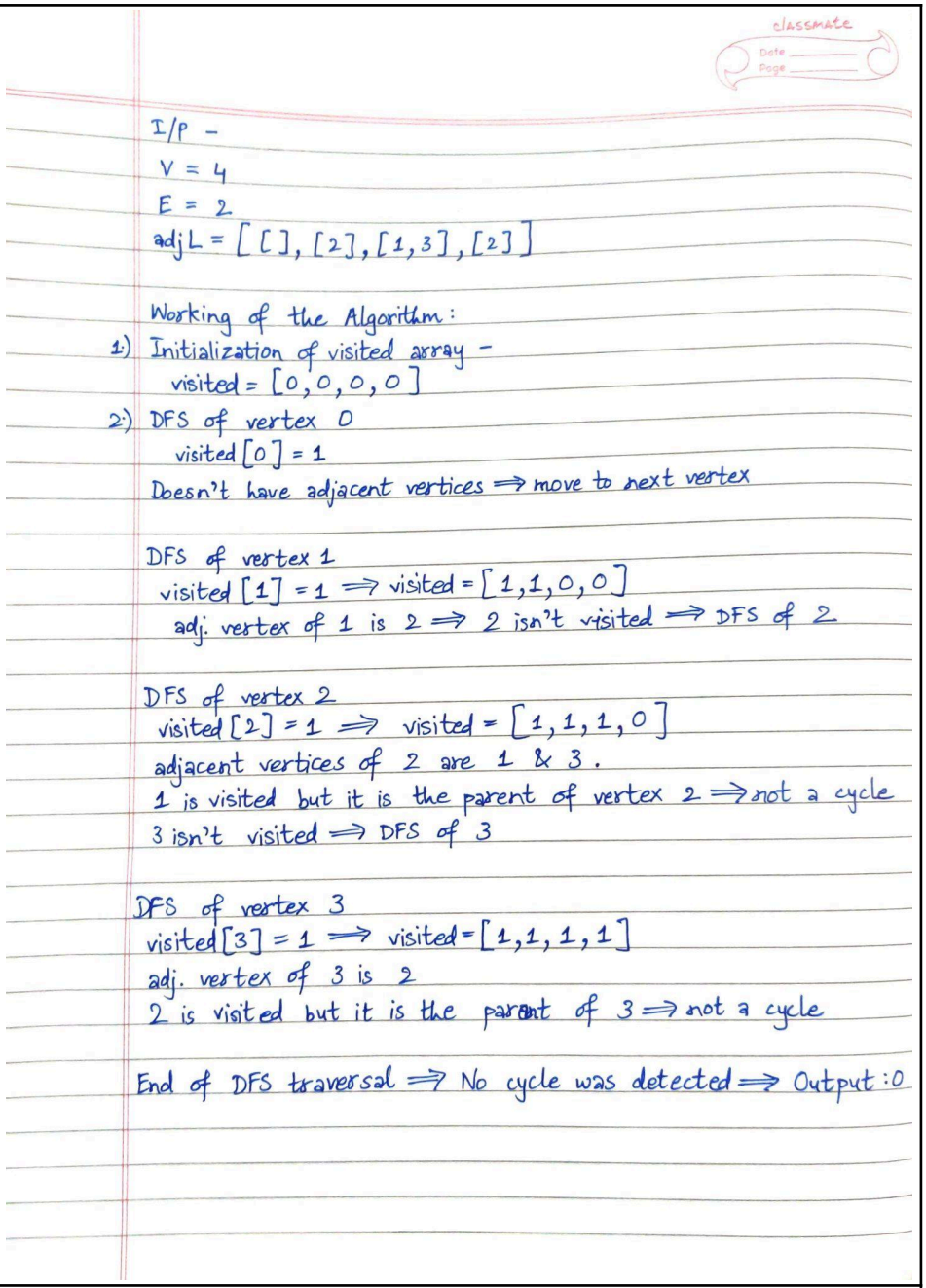
Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.
Department of AI-ML

Experiment	8
Aim	Implement the given problem statement
Objective	Application of graph traversal techniques Given an undirected graph with V vertices labelled from 0 to V-1 and E edges, check whether it contains any cycle or not. Graph is in the form of adjacency list where adj[i] contains all the nodes that the ith node is having edges with. NOTE: The adjacency list denotes the edges of the graph where edges[i] stores all other vertices to which ith vertex is connected.
Name	Balla Mahadev Shrikrishna
UCID	2023300010
Class	A
Batch	A
Date of Submission	17-10-24

Explanation of the technique used	

Program(Code)	#include <stdio.h>
	#include <stdlib.h>
	
	typedef struct Node{
	int val;
	struct Node* next;
	}Node;
	
	Node* createNode(int val){
	Node* new = (Node*)malloc(sizeof(Node));
	new->val = val;
	new->next = NULL;
	return new;
	}

```
typedef struct Graph{
    int v;
    Node** adjL;
}Graph;

Graph *createGraph(int v){
    Graph *g = (Graph *)malloc(sizeof(Graph));
    g->v = v;
    g->adjL = malloc(v * sizeof(struct Node*));
    for(int i=0; i<v; i++){
        g->adjL[i] = NULL;
    }
    return g;
}

void addEdge(Graph *g, int s, int t){
    Node *new = createNode(t);
    new->next = g->adjL[s];
    g->adjL[s] = new;

    new = createNode(s);
    new->next = g->adjL[t];
    g->adjL[t] = new;
}

void printGraph(Graph* g) {
    for (int i = 0; i < g->v; i++) {
        Node* temp = g->adjL[i];
        printf("Vertex %d : ", i);
        while (temp) {
            printf("%d -> ", temp->val);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

int DFS(Graph* g, int v, int visited[], int parent) {
    visited[v] = 1;
    Node* temp = g->adjL[v];
    while(temp!=NULL){
        if(!visited[temp->val]){
            if(DFS(g, temp->val, visited, v)){
                return 1;
            }
        }
        else if(temp->val!=parent){
            return 1;
        }
```

```
        temp = temp->next;
    }
    return 0;
}

int detectCycle(Graph* g){
    int* visited = (int*)malloc(g->v * sizeof(int));
    for(int i=0; i<g->v; i++){
        visited[i] = 0;
    }

    for(int i=0; i<g->v; i++){
        if(!visited[i]){
            if(DFS(g, i, visited, -1)){
                free(visited);
                return 1;
            }
        }
    }
    free(visited);
    return 0;
}

int main(){
    int v=0, e=0, s=0, t=0;

    printf("Enter the number of vertices : ");
    scanf("%d", &v);

    Graph* graph = createGraph(v);

    printf("Enter the number of edges : ");
    scanf("%d", &e);

    for(int i=0; i<e; i++){
        printf("Enter edge %d vertices : ", (i+1));
        scanf("%d %d", &s, &t);
        addEdge(graph, s, t);
    }
    printf("Adjacency List -\n");
    printGraph(graph);
    printf("O/P : ");
    if(detectCycle(graph)){
        printf("1\n");
    }
    else{
        printf("0\n");
    }

    free(graph->adjL);
```

```
    free(graph);
    return 0;
}
```

Output

graphs.c

graphs.c > main()

```
88  int main(){
89      int v=0, e=0, s=0, t=0;
90
91      printf("Enter the number of vertices : ");
92      scanf("%d", &v);
93
94      Graph* graph = createGraph(v);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Mahadev\S.E\DS\Lab Sessions> gcc graphs.c
PS C:\Mahadev\S.E\DS\Lab Sessions> ./a.exe
Enter the number of vertices : 5
Enter the number of edges : 5
Enter edge 1 vertices : 0 1
Enter edge 2 vertices : 1 2
Enter edge 3 vertices : 1 4
Enter edge 4 vertices : 2 3
Enter edge 5 vertices : 3 4
Adjacency List -
Vertex 0 : 1 -> NULL
Vertex 1 : 4 -> 2 -> 0 -> NULL
Vertex 2 : 3 -> 1 -> NULL
Vertex 3 : 4 -> 2 -> NULL
Vertex 4 : 3 -> 1 -> NULL
O/P : 1
PS C:\Mahadev\S.E\DS\Lab Sessions> ./a.exe
Enter the number of vertices : 4
Enter the number of edges : 2
Enter edge 1 vertices : 1 2
Enter edge 2 vertices : 2 3
Adjacency List -
Vertex 0 : NULL
Vertex 1 : 2 -> NULL
Vertex 2 : 3 -> 1 -> NULL
Vertex 3 : 2 -> NULL
O/P : 0
PS C:\Mahadev\S.E\DS\Lab Sessions>
```

Conclusion

In this experiment, we implemented a solution to detect cycles in an undirected graph using Depth-First Search (DFS). The graph was represented using an adjacency list, making the structure efficient for traversal. Through the DFS traversal, we explored the vertices and their connections, marking nodes as visited. If a node was revisited, and it was not the parent node, this indicated the presence of a cycle. The algorithm efficiently identified whether a cycle existed, based on these conditions.