Lab -

DIP : Design and Analysis of Algorithm

AIM : Write a program to solve the following Algorithm

1. FIND TOPOLOGICAL SORT FROM GIVEN GRAPH USING DFS

2. FIND TOPOLOGICAL SORT FROM GIVEN GRAPH USING SOURCE REMOVAL METHOD

3. FIND ARTICULATION POINT FROM GIVEN GRAPH USING DFS

Name : Ambalia Harshit

Department : Computer Engineering (M.Tech sem II)

Roll no : MT001

Date : Jan 2024

Program 01 : FIND TOPOLOGICAL SORT FROM GIVEN GRAPH USING DFS

- Code :

```c
// Topological sort using DFS

#include <stdio.h>

void dfs(int start, int length, int graph[length][length], int
global_visited[length]) {
   int visited[length];
   for( int i=0;i<length;i++ ) {
       visited[i] = global_visited[i];
   }
   int stack[(length*(length-1))/2];
   int top = -1;
   top++;
   stack[top] = start;
   visited[start] = 1;
   while( top!=-1 ) {
       int current = stack[top];
       printf("Visiting node : %d\n", current+1);
       top--;
       for( int i=0;i<length;i++ ) {
           if (graph[current][i] && !visited[i]) {
               top++;
               stack[top] = i; // Add adjecent nodes to stack
               visited[i] = 1; // Node is now visited
               printf("Adding node %d to stack\n", i+1);
           }
       }
   }
   for( int i=0;i<length;i++ ) {
       global_visited[i] = visited[i];
   }
}

void topological_sort_using_dfs(int length, int graph[length][length]) {
   int global_visited[length];
```

```
    for( int i=0;i<length;i++ )
        global_visited[i] = 0;
    for( int i=0;i<length;i++ ) {
        if( global_visited[i]==0 ) {
            printf("\nStarting DFS from node %d\n", i+1);
            dfs(i, length, graph, global_visited);
        }
    }
}

int main() {
    int n = 4;
    int graph[4][4] = { {0, 1, 0, 0},
                        {0, 0, 0, 1},
                        {0, 1, 0, 1},
                        {0, 0, 0, 0} };
    int global_visited[4] = {0};
    int visited[4] = {0};
    topological_sort_using_dfs(n, graph);
}
```

- Output Screen-shots / Tracing :

```
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_07$ cd "/ho
01.c -o ex01 && "/home/hr/Documents/Semester_10/Lab_DAA/

Starting DFS from node 1
Visiting node : 1
Adding node 2 to stack
Visiting node : 2
Adding node 4 to stack
Visiting node : 4

Starting DFS from node 3
Visiting node : 3
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_07$
```

Program 02 : FIND TOPOLOGICAL SORT FROM GIVEN GRAPH USING SOURCE REMOVAL METHOD

- Code :

```
// Topological sort using source removal

#include <stdio.h>

void find_indegree(int length, int graph[length][length], int
indegree[length]) {
   for( int i=0;i<length;i++ ) {
       for( int j=0;j<length;j++ ) {
           if( graph[i][j]==1 ) {
               indegree[j]++;
           }
       }
   }
}

void deleted_node(int length, int node, int indegree[length], int
graph[length][length]) {
   for( int i=0;i<length;i++ ) {
       if(graph[node][i]==1) {
           indegree[i]--;
       }
   }
}

void topological_sort_using_source_removal(int length, int
indegree[length], int graph[length][length]) {
   int visited[length];
   for( int i=0;i<length;i++ ) {
       visited[i] = 0;
   }

   while(1) {
       int temp = 0;
       for( int i=0;i<length;i++ ) {
           if(indegree[i]==0 && visited[i]==0){
```

```
                printf("deleting node : %d\n", i+1);
                deleted_node(length, i, indegree, graph);
                visited[i] = 1;
                temp = 1;
            }
        }
        if(temp==0) {
            break;
        }
    }
}

int main() {
    int n = 4;
    int graph[4][4] = { {0, 1, 0, 0},
                        {0, 0, 0, 1},
                        {0, 1, 0, 1},
                        {0, 0, 0, 0} };
    int indegree[4] = {0};
    find_indegree(n, graph, indegree);
    for( int i=0;i<n;i++ ) {
        printf("in degree of %d is %d\n", i, indegree[i]);
    }
    topological_sort_using_source_removal(n, indegree, graph);
}
```

- Output Screen-shots / Tracing :

```
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_07$ cd "/ho
02.c -o ex02 && "/home/hr/Documents/Semester_10/Lab_DAA/
in degree of 0 is 0
in degree of 1 is 2
in degree of 2 is 0
in degree of 3 is 2
deleting node : 1
deleting node : 3
deleting node : 2
deleting node : 4
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_07$
```

Program 03 : FIND ARTICULATION POINT FROM GIVEN GRAPH USING DFS

- Code :

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

int min(int a, int b) {
   return (a<b) ? a:b;
}

void DFS(int length, int **graph, int node, int visited[], int
disc_time[], int lowest_disc_time[], int parent[], int is_ap[]) {
   static int time = 0;
   int children = 0;
   visited[node] = 1;
   disc_time[node] = lowest_disc_time[node] = ++time;

   printf("\nVisiting node %d\n", node);
   printf("Updated disc_time[%d] = %d\n", node, disc_time[node]);
   printf("Updated lowest_disc_time[%d] = %d\n", node,
lowest_disc_time[node]);

   for( int v=0;v<length;v++ ) {
       if( graph[node][v] ) {
           if ( !visited[v] ) {
               children++;
               parent[v] = node;
               printf("DFS from %d to %d\n", node, v);
               DFS(length, graph, v, visited, disc_time, lowest_disc_time,
parent, is_ap);
               printf("Returned from DFS from %d to %d\n", node, v);
               lowest_disc_time[node] = min(lowest_disc_time[node],
lowest_disc_time[v]);
               printf("Updated lowest_disc_time[%d] = %d\n", node,
lowest_disc_time[node]);
               if( parent[node]==-1 && children>1 ) {
                   is_ap[node] = 1;
                   printf("Articulation point found: %d\n", node);
```

```
                }
                if( parent[node]!=-1 &&
lowest_disc_time[v]>=disc_time[node] ) {
                    is_ap[node] = 1;
                    printf("Articulation point found: %d\n", node);
                }
            }
            else if( v!=parent[node] ) {
                lowest_disc_time[node] = min(lowest_disc_time[node],
disc_time[v]);
                printf("Back edge found between %d and %d\n", node, v);
                printf("Updated lowest_disc_time[%d] = %d\n", node,
lowest_disc_time[node]);
            }
        }
    }
}

void my_articulation_point(int length, int **graph, int V, int is_ap[]) {
    int visited[length];
    int disc_time[length];
    int lowest_disc_time[length];
    int parent[length];

    for( int i=0;i<V;i++ ) {
        parent[i] = -1;
        visited[i] = false;
        is_ap[i] = false;
    }

    for( int i=0;i<V;i++ )
        if (!visited[i])
            DFS(length, graph, i, visited, disc_time, lowest_disc_time,
parent, is_ap);
}

int main() {
    int V = 9;

    int **graph = (int **)malloc(V * sizeof(int *));
```

```
    for (int i = 0; i < V; i++)
        graph[i] = (int *)malloc(V * sizeof(int));

    // Populate the adjacency matrix
    int adj_matrix[9][9] = {
        {0, 1, 0, 1, 0, 0, 0, 0, 0},
        {1, 0, 1, 0, 0, 0, 0, 0, 0},
        {0, 1, 0, 1, 0, 0, 0, 1, 1},
        {1, 0, 1, 0, 1, 0, 0, 0, 0},
        {0, 0, 0, 1, 0, 1, 1, 0, 0},
        {0, 0, 0, 0, 1, 0, 0, 0, 0},
        {0, 0, 0, 0, 1, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 0, 0, 0},
    };

    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            graph[i][j] = adj_matrix[i][j];

    int is_ap[V];

    my_articulation_point(V, graph, V, is_ap);

    printf("Articulation Points: ");
    for( int i=0;i<V;i++ )
        if( is_ap[i] )
            printf("%d ", i + 1);
    printf("\n");

    return 0;
}
```

- Output Screen-shots / Tracing :

```
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_07$ cd "/ho
03.c -o ex03 && "/home/hr/Documents/Semester_10/Lab_DAA/

Visiting node 0
Updated disc_time[0] = 1
Updated lowest_disc_time[0] = 1
DFS from 0 to 1

Visiting node 1
Updated disc_time[1] = 2
Updated lowest_disc_time[1] = 2
DFS from 1 to 2

Visiting node 2
Updated disc_time[2] = 3
Updated lowest_disc_time[2] = 3
DFS from 2 to 3

Visiting node 3
Updated disc_time[3] = 4
Updated lowest_disc_time[3] = 4
Back edge found between 3 and 0
Updated lowest_disc_time[3] = 1
DFS from 3 to 4

Visiting node 4
Updated disc_time[4] = 5
Updated lowest_disc_time[4] = 5
DFS from 4 to 5

Visiting node 5
Updated disc_time[5] = 6
Updated lowest_disc_time[5] = 6
Returned from DFS from 4 to 5
```

```
Visiting node 5
Updated disc_time[5] = 6
Updated lowest_disc_time[5] = 6
Returned from DFS from 4 to 5
Updated lowest_disc_time[4] = 5
Articulation point found: 4
DFS from 4 to 6

Visiting node 6
Updated disc_time[6] = 7
Updated lowest_disc_time[6] = 7
Returned from DFS from 4 to 6
Updated lowest_disc_time[4] = 5
Articulation point found: 4
Returned from DFS from 3 to 4
Updated lowest_disc_time[3] = 1
Articulation point found: 3
Returned from DFS from 2 to 3
Updated lowest_disc_time[2] = 1
DFS from 2 to 7

Visiting node 7
Updated disc_time[7] = 8
Updated lowest_disc_time[7] = 8
Returned from DFS from 2 to 7
Updated lowest_disc_time[2] = 1
Articulation point found: 2
DFS from 2 to 8
```

```
Visiting node 8
Updated disc_time[8] = 9
Updated lowest_disc_time[8] = 9
Returned from DFS from 2 to 8
Updated lowest_disc_time[2] = 1
Articulation point found: 2
Returned from DFS from 1 to 2
Updated lowest_disc_time[1] = 1
Returned from DFS from 0 to 1
Updated lowest_disc_time[0] = 1
Back edge found between 0 and 3
Updated lowest_disc_time[0] = 1
Articulation Points: 3 4 5
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_07$
```