Ex. No. 9b **Depth First Search**

Date:

**Aim**

To create adjacency matrix of the given graph and to perform depth first search traversal.

**Algorithm**

1. Start

2. Obtain Adjacency matrix for the given graph

3. Define a Stack of size total number of vertices in the graph.

4. Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.

5. Visit any one of the adjacent vertex of the verex which is at top of the stack which is not visited and push it on to the stack.

6. Repeat step 5 until there are no new vertex to be visit from the vertex on top of the stack.

7. When there is no new vertex to be visit then use back tracking and pop one vertex from the stack.

8. Repeat steps 5, 6 and 7 until stack becomes Empty.

9. When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph.

10. Stop

**Program**

/\* DFS on undirected graph \*/

#include <stdio.h>

#include <stdlib.h>

#define true 1

#define false 0

#define MAX 5

struct Vertex

{

char label;

int visited;

};

int stack[MAX];

int top = -1;

struct Vertex\* lstVertices[MAX];

static int adjMatrix[MAX][MAX];

int vertexCount = 0;

void push(int item)

{

stack[++top] = item;

}

int pop()

{

return stack[top--];

}

int peek()

{

return stack[top];

}

int isStackEmpty()

{

return top == -1;

}

void addVertex(char label)

{

struct Vertex\* vertex = (struct Vertex\*)

malloc(sizeof(struct Vertex));

vertex->label = label;

vertex->visited = false;

lstVertices[vertexCount++] = vertex;

}

void addEdge(int start, int end)

{

adjMatrix[start][end] = 1;

adjMatrix[end][start] = 1;

}

void displayVertex(int vertexIndex)

{

printf("%c ", lstVertices[vertexIndex]->label);

}

int getAdjUnvisitedVertex(int vertexIndex)

{

int i;

for(i = 0; i < vertexCount; i++)

{

if(adjMatrix[vertexIndex][i] == 1 &&

lstVertices[i]->visited == false)

return i;

}

return -1;

}

void depthFirstSearch()

{

int i;

lstVertices[0]->visited = true;

displayVertex(0);

push(0);

while(!isStackEmpty())

{

int unvisitedVertex = getAdjUnvisitedVertex(peek());

if(unvisitedVertex == -1)

pop();

else

{

lstVertices[unvisitedVertex]->visited = true;

displayVertex(unvisitedVertex);

push(unvisitedVertex);

}

}

for(i = 0;i < vertexCount;i++)

lstVertices[i]->visited = false;

}

void main()

{

int i, j, n, edges, orgn, destn;

char ch;

printf("Enter no. of vertices : ");

scanf("%d", &n);

edges = n \* (n - 1);

printf("Enter Vertex Labels : \n");

for (i=0; i<n; i++)

{

fflush(stdin);

scanf("%c", &ch);

addVertex(ch);

}

for(i=0; i<edges; i++)

{

printf("Enter edge ( -1 -1 to quit ) : ");

scanf("%d %d", &orgn, &destn);

if((orgn == -1) && (destn == -1))

break;

if(orgn>=n || destn>=n || orgn<0 || destn<0)

printf("Invalid edge!\n");

else

addEdge(orgn, destn);

}

printf("\nDepth First Search: ");

depthFirstSearch();

}

**Output**

**Result**

Thus depth first traversal is executed on the given undirected graph.