Department of
Computer Science and Engineering

Title: Implement Depth-First Search Traversal

Data Structures Lab
CSE 106

Green University of Bangladesh

1 Objective(s)

- To understand how to represent a graph using adjacency list.
- To understand how Depth-First Search (DFS) works.

2 Problem analysis

Two of the most popular tree traversal algorithms are breadth-first search (BFS) and depth-first search (DFS). Both methods visit all vertices and edges of a graph; however, they are different in the way in which they perform the traversal. This difference determines which of the two algorithms is better suited for a specific purpose.

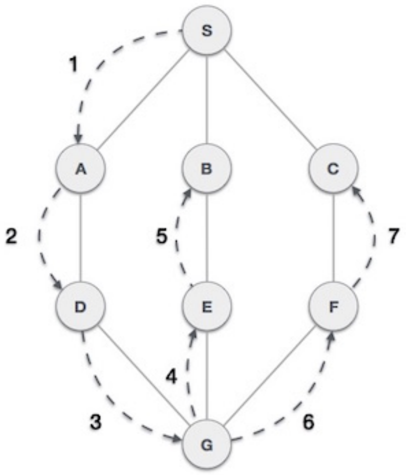
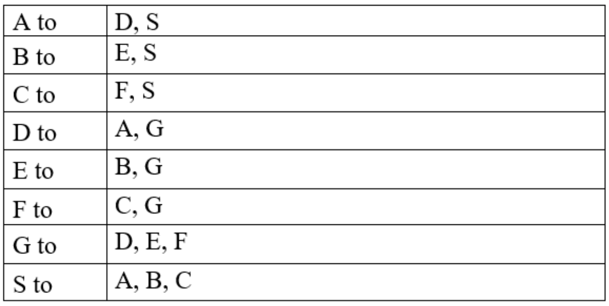


Figure 1: A simple graph

Adjacency List:
Vertices are labelled (or re-labelled) from 0 to V(G)- 1. Corresponding to each vertex is a list (either an array or linked list) of its neighbours. Table: 1 represents the adjacency list of figure1.



A to	D, S
B to	E, S
C to	F, S
D to	A, G
E to	B, G
F to	C, G
G to	D, E, F
S to	A, B, C

Table:1

DFS:
Depth-first Search or Depth-first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. Traversal means visiting all the nodes of a graph. Figure 1 shows the DFS graph traversal.As in the example given above, the DFS algorithm traverses from S to A to D to G to E to B first. Now from B there is an edge to S, but it is not possible to visit S as it was already visited. Now B will return to its parent E and E will return to G. From G it has another child F, so traversal happened on F and after that F to C to complete total traversal.

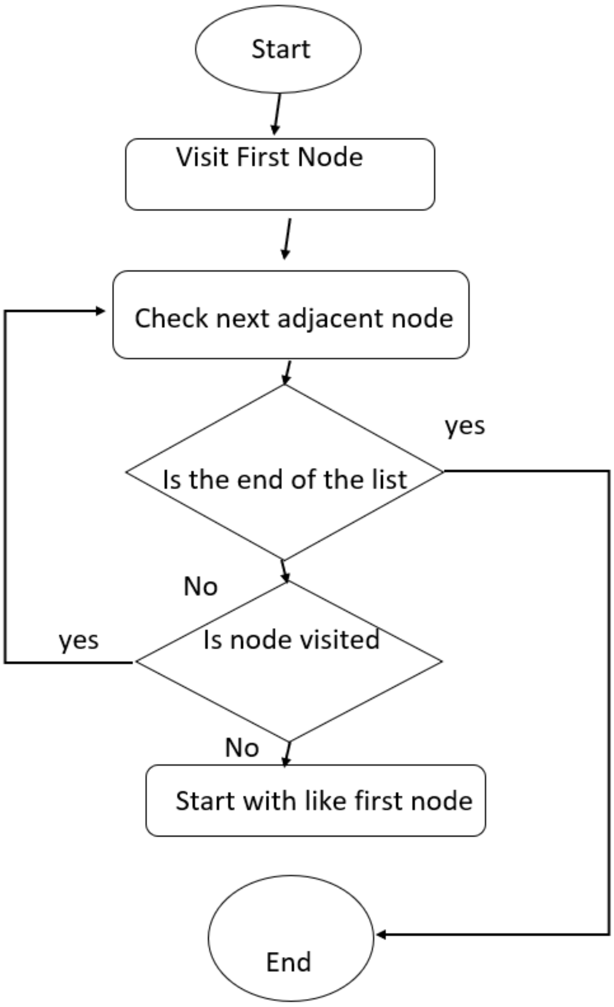
3 Algorithm (DFS)

A standard DFS implementation puts each vertex of the graph into one of two categories:
1. Visited
2. Not Visited
The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
The DFS algorithm works as follows:

Algorithm 1: Depth-First Search

```
Data: graph[][], color[], prev[], d[], f[], time
Function DFS():
    for each vertex $u \in V - \{s\}$ do
        $color[u] = WHITE$
        $prev[u] = null$
        $f[u] = inf$
        $d[u] = inf$
    end
    $time = 0$
    for each $u \in V$ do
        if $color[u] == WHITE$ then
            $DFS\_Visit(u)$
        end
    end
return
Function DFS_ Visit(u):
    $color[u] = GRAY$
    $time = time + 1$
    $d[u] = time$
    for each $v \in adj[u]$ do
        if $color[v] == WHITE$ then
            $prev[v] = u$
            $DFS\_Visit(v)$
        end
    end
    $color[u] = BLACK$
    $time = time + 1$
    $f[u] = time$
return
```

4 Flowchart



5 Implementation in C

```
// DFS algorithm in C

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

struct node {
  int vertex;
  struct node* next;
};

struct node* createNode(int v);

struct Graph {
  int numVertices;
  int* visited;

  // We need int** to store a two dimensional array.
  // Similary, we need struct node** to store an array of Linked lists
  struct node** adjLists;
```

```c
};

// DFS algo
void DFS(struct Graph* graph, int vertex) {
  struct node* adjList = graph->adjLists[vertex];
  struct node* temp = adjList;

  graph->visited[vertex] = 1;
  printf("Visited %d \n", vertex);

  while (temp != NULL) {
    int connectedVertex = temp->vertex;

    if (graph->visited[connectedVertex] == 0) {
      DFS(graph, connectedVertex);
    }
    temp = temp->next;
  }
}

// Create a node
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}

// Create graph
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));

  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }
  return graph;
}

// Add edge
void addEdge(struct Graph* graph, int src, int dest) {
  // Add edge from src to dest
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  // Add edge from dest to src
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}

```

```c
// Print the graph
void printGraph(struct Graph* graph) {
  int v;
  for (v = 0; v < graph->numVertices; v++) {
    struct node* temp = graph->adjLists[v];
    printf("\n Adjacency list of vertex %d\n ", v);
    while (temp) {
      printf("%d -> ", temp->vertex);
      temp = temp->next;
    }
    printf("\n");
  }
}

int main() {
  struct Graph* graph = createGraph(4);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 2);
  addEdge(graph, 1, 2);
  addEdge(graph, 2, 3);

  printGraph(graph);

  DFS(graph, 2);

  return 0;
}
```

6 Sample Input/Output (Compilation, Debugging & Testing)

Output:
S A D G E B F C

7 Lab Task (Please implement yourself and show the output to the instructor)

1. Write a program to perform DFS traversal by taking user input of a graph.
2. Write a program to find the path from source to destination using DFS.

8 Lab Exercise (Submit as a report)

- Write a program to find the topological order of nodes of a graph using DFS.

9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.