Experiment No. 9: Depth First Search and Breath First Search

Name : Kaushal Tare
Rol No. / Div : 50 / Comps 3

Aim : Implementation of DFS and BFS traversal of graph.

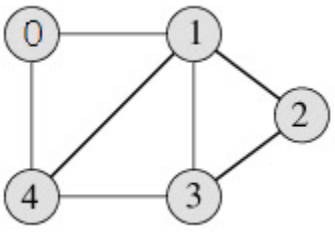
Objective:

1. Understand the Graph data structure and its basic operations.
2. Understand the method of representing a graph.
3. Understand the method of constructing the Graph ADT and defining its operations

Theory:

A graph is a collection of nodes or vertex, connected in pairs by lines referred as edges. A graph can be directed or undirected graph.

One method of traversing through nodes is depth first search. Here we traverse from starting node and proceeds from top to bottom. At a moment we reach a dead end from where the further movement is not possible and we backtrack and then proceed according to left right order. A stack is used to keep track of a visited node which helps in backtracking.



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

DFS Traversal –0 1 2 3 4

Algorithm

Algorithm: DFS_LL(V)

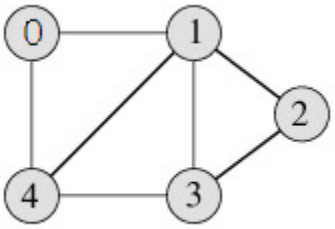
Input: V is a starting vertex

Output : A list VISIT giving order of visited vertices during traversal.

Description: linked structure of graph with gptr as pointer

1. if gptr = NULL then
   print "Graph is empty" exit
2. u=v
3. OPEN.PUSH(u)
4. while OPEN.TOP !=NULL do
   u=OPEN.POP()
   if search(VISIT,u) = FALSE then
   INSERT_END(VISIT,u)
   Ptr = gptr(u)
   While ptr.LINK != NULL do
   Vptr = ptr.LINK
   OPEN.PUSH(vptr.LABEL)
   End while
   End if
   End while
5. Return VISIT
6. Stop

BFS Traversal



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

BFS Traversal – 0 1 4 2 3

Algorithm

Algorithm: DFS()

i=0

count=1

visited[i]=1

print("Visited vertex i")

repeat this till queue is empty or all nodes visited

repeat this for all nodes from first till last

if(g[i][j]!=0&&visited[j]!=1)

{

push(j)

}

i=pop()

print("Visited vertex i")

visited[i]=1

count++

Algorithm: BFS()

i=0

count=1

visited[i]=1

print("Visited vertex i")

repeat this till queue is empty or all nodes visited

repeat this for all nodes from first till last

if(g[i][j]!=0&&visited[j]!=1)

{

enqueue(j)

}

i=dequeue()

print("Visited vertex i")

visited[i]=1

count++

Code:

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

// Define the maximum number of vertices in the graph
#define MAX_VERTICES 100

// Define a stack for DFS and a queue for BFS
int stack[MAX_VERTICES];
int top = -1;
int queue[MAX_VERTICES];
int front = -1, rear = -1;

int visited[MAX_VERTICES];

// Define the adjacency matrix for the graph
int graph[MAX_VERTICES][MAX_VERTICES];
int numVertices;

void push(int vertex) {
    stack[++top] = vertex;
}

int pop() {
    return stack[top--];
}

void enqueue(int vertex) {
    if (rear == MAX_VERTICES - 1)
        printf("Queue is full.\n");
```

```
    else
        queue[++rear] = vertex;
}

int dequeue() {
    if (front == rear)
        return -1; // Empty queue
    return queue[++front];
}

void initGraph(int vertices) {
    numVertices = vertices;
    for (int i = 0; i < numVertices; i++) {
        visited[i] = 0;
        for (int j = 0; j < numVertices; j++) {
            graph[i][j] = 0;
        }
    }
}

void addEdge(int start, int end) {
    graph[start][end] = 1;
}

void DFS(int vertex) {
    int u, v;
    push(vertex);

    while (top != -1) {
        u = pop();
        if (visited[u] == 0) {
            printf("Visited vertex %d\n", u);
            visited[u] = 1;
            for (v = 0; v < numVertices; v++) {
                if (graph[u][v] && !visited[v]) {
                    push(v);
                }
            }
        }
    }
}
```

```
void BFS(int vertex) {
    int u, v;
    enqueue(vertex);

    while (front != rear) {
        u = dequeue();
        if (visited[u] == 0) {
            printf("Visited vertex %d\n", u);
            visited[u] = 1;
            for (v = 0; v < numVertices; v++) {
                if (graph[u][v] && !visited[v]) {
                    enqueue(v);
                }
            }
        }
    }
}

int main() {
    int vertices, edges, startVertex;

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

    printf("Enter the edges (format: start end):\n");
    for (int i = 0; i < edges; i++) {
        int start, end;
        scanf("%d %d", &start, &end);
        addEdge(start, end);
    }

    printf("Enter the starting vertex for traversal: ");
    scanf("%d", &startVertex);

    printf("DFS Traversal:\n");
    DFS(startVertex);

    // Reset visited array
    for (int i = 0; i < vertices; i++) {
```

```
    visited[i] = 0;
  }

  printf("\nBFS Traversal:\n");
  BFS(startVertex);

  return 0;
}
```

Output:

```
Enter the number of vertices and edges: 4 5
Enter the edges (format: start end):
2 3
3 4
4 5
5 6
6 7
Enter the starting vertex for traversal: 2
DFS Traversal:
Visited vertex 2
Visited vertex 3

BFS Traversal:
Visited vertex 2
Visited vertex 3
```

Conclusion:

DFS (Depth-First Search): It explores a graph by delving as deep as possible along each branch before backtracking. It is useful for discovering connected components within a graph.

BFS (Breadth-First Search): This method explores a graph level by level, making it suitable for finding the shortest path and exploring neighbors.

Both DFS and BFS are indispensable tools for various computer science applications, from pathfinding to network analysis and graph-related problem solving. The choice between them depends on the specific problem and objectives.