Data Structures Odyssey: Exploring the Foundations of Computing

Ex. No.:12	Graph Traversal	Date:09/05/2024

Write a C program to create a graph and perform a Breadth First Search and Depth First Search.

Algorithm:

DFS

1)Start with an empty stack and a set to track visited nodes.
2) Push the starting node onto the stack and mark it as visited.
3) While the stack is not empty, do the following:
4) If the stack is not empty, pop a node from the stack.
Process the node.
5) For each unvisited neighbor of the node, push the neighbor onto the stack and mark it as visited.
6) Repeat steps 3-6 until the stack is empty.
7) Stop

BFS

1) Start with an empty queue and a set to track visited nodes.
2)Enqueue the starting node into the queue and mark it as visited.
3) While the queue is not empty, do the following:
4) If the queue is not empty, dequeue a node from the queue.
Process the node.
5) For each unvisited neighbor of the node, enqueue the neighbor into the queue and mark it as visited.
6) Repeat steps 3-6 until the queue is empty.
7) Stop

PROGRAM: BFS

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

struct node { int
vertex; struct
node* next;

}; struct adj_list {
struct node* head;
};

struct graph { int
num_vertices; struct
adj_list* adj_lists; int*
visited;
};
struct node* new_node(int vertex) { struct node* new_node =
(struct node*)malloc(sizeof(struct node)); new_node->vertex =
vertex; new_node->next = NULL; return new_node;
}
struct graph* create_graph(int n) { struct graph* graph = (struct
graph*)malloc(sizeof(struct graph)); graph->num_vertices = n;
graph->adj_lists = (struct adj_list*)malloc(n * sizeof(struct adj_list));
graph->visited = (int*)malloc(n * sizeof(int));



int i; for (i = 0; i< n; i++) { graph-
>adj_lists[i].head = NULL;
graph->visited[i] = 0;
}
```

```
return graph;
}
void add_edge(struct graph* graph, int src, int dest) {
struct node* new_node1 = new_node(dest);
new_node1->next = graph->adj_lists[src].head;
graph->adj_lists[src].head = new_node1; struct
node* new_node2 = new_node(src); new_node2-
>next = graph->adj_lists[dest].head; graph-
>adj_lists[dest].head = new_node2;
}
void bfs(struct graph* graph, int v) { int queue[1000]; int
front = -1; int rear = -1; graph->visited[v] = 1;
queue[++rear] = v; while (front != rear) { int
current_vertex = queue[++front]; printf("%d ",
current_vertex); struct node* temp = graph-
>adj_lists[current_vertex].head; while (temp != NULL) { int
adj_vertex = temp->vertex;




if (graph->visited[adj_vertex] == 0) { graph-
>visited[adj_vertex] = 1; queue[++rear] =
adj_vertex;
}
temp = temp->next;
}
}
}

int main() { struct graph* graph =
create_graph(6); add_edge(graph, 0, 1);
add_edge(graph, 0, 2); add_edge(graph, 1, 3);
```

```
add_edge(graph, 1, 4); add_edge(graph, 2, 4);
add_edge(graph, 3, 4); add_edge(graph, 3, 5);
add_edge(graph, 4,5); printf("BFS traversal
starting from vertex 0: "); bfs(graph, 0);

return 0;
}
```

DFS:

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

// Globally declared visited array int
vis[100];


struct Graph {
    int V;
int E;
int** Adj;
};

struct Graph* adjMatrix()
{
    struct Graph* G = (struct Graph*)
malloc(sizeof(struct Graph));     if
(!G) {         printf("Memory Error\n");
return NULL;
    }
    G->V = 7;
    G->E = 7;
```

```
    G->Adj = (int**)malloc((G->V) * sizeof(int*));
for (int k = 0; k < G->V; k++) {
        G->Adj[k] = (int*)malloc((G->V) * sizeof(int));
    }

    for (int u = 0; u < G->V; u++) {
for (int v = 0; v < G->V; v++) {
            G->Adj[u][v] = 0;
        }
    }
    G->Adj[0][1] = G->Adj[1][0] = 1;
    G->Adj[0][2] = G->Adj[2][0] = 1;
G->Adj[1][3] = G->Adj[3][1] = 1;
G->Adj[1][4] = G->Adj[4][1] = 1;
G->Adj[1][5] = G->Adj[5][1] = 1;
     G->Adj[1][6] = G->Adj[6][1] = 1;     G-
         >Adj[6][2] = G->Adj[2][6] = 1;

    return G;
}
 void DFS(struct Graph* G, int u)

{     vis[u] = 1;     printf("%d ", u);
for (int v = 0; v < G->V; v++) {
if (!vis[v] && G->Adj[u][v]) {
            DFS(G, v);
        }
    }
}

void DFStraversal(struct Graph* G)
{     for (int i = 0; i < 100; i++)
{         vis[i] = 0;
```

```
    }
    for (int i = 0; i < G->V; i++) {
if (!vis[i]) {
            DFS(G, i);
        }
    }
}
 void main()
{


 struct Graph* G;
    G = adjMatrix();
    DFStraversal(G);
}
```

OUTPUT:

```
BFS traversal starting from vertex 0: 0 2 1 4 3 5 
0 1 3 4 5 6 2
```

RESULT: Thus, the program was successfully executed.