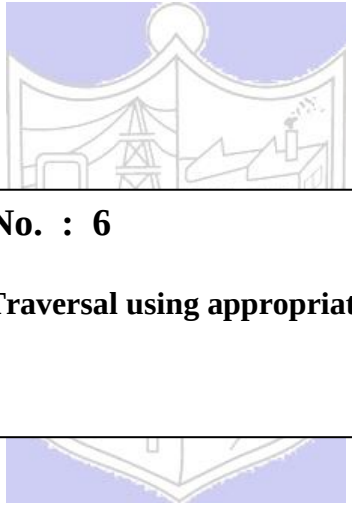


Experiment No. : 6

Title: Graph Traversal using appropriate data structure

Batch: B1 **Roll No.: 1914078** **Experiment No.: 6**

Aim: Implement a menu driven program in to accept a graph data from the user and traverse it using BFS technique.

Resources Used: C/ C++ editor and compiler.

Theory:

Graph

Given an undirected graph G= (V,E) and a vertex V in V(G), then we are interested in visiting all vertices in G that are reachable from V i.e. all vertices connected to V. There are two techniques of doing it namely Depth First Search (DFS) and Breadth First Search(BFS).

Depth First Search

The procedure of performing DFS on an undirected graph can be as follows :
The starting vertex v is visited. Next an unvisited vertex w adjacent to v is selected and a depth first search from w is initiated. When a vertex u is reached such that all its adjacent vertices have been visited, we back up to the last vertex visited which has an unvisited vertex w adjacent to it and initiate a depth first search from w. the search terminates when no unvisited vertex can be reached from any of the visited ones.
Given an undirected graph G=(V,E) with n vertices and an array visited[n] initially set to false, this algorithm, dfs (v) visits all vertices reachable from v. Visited is a global array.

Breadth First Search

Starting at vertex v and making it as visited, BFS visits next all unvisited vertices adjacent to v. then unvisited vertices adjacent to there vertices are visited and so on.

A breadth first search of G is carried out beginning at vertex v as bfs (v). All vertices visited are marked as visited [i]=true. The graph G and arrat visited are global and visited is initialized to false. Initialize, addqueue, emptyqueue, deletequeue are the functions to handle operations on queue.

Algorithm :

Implement the queue ADT, Represent the graph using adjacency matrix and implement following pseudo code for BFS.

Pseudo Code: bfs (v)

initialize queue q

visited [v] = true

addqueue(q,v)

while not emptyqueue

for all vertices w adjacent to v do

if not visited [w] then

addqueue (q,w)

visited [w]=true

deletequeue

Results:
A program depicting the BFS using adjacency matrix and capable of handling all possible boundary conditions and the same is reflected clearly in the output.

Code:-

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

struct node
{
    int data;
    struct node *next;
} *front = NULL, *rear = NULL;

int total_elements;

// declaration
void menu(int **, int[]);
void push(int);
void enqueue(int);
void pop();
int check_visited(int, int, int[]);
void bfs(int **, int[], int);
void dfs(int **, int[], int);

void main()
{
    system("cls");
    char choice;
    printf("--------- Enter Graph Values --------------\n");
    printf("Enter the no. of nodes: ");
```

```
    scanf("%d", &total_elements);
    int values_at_nodes[total_elements];
    int **matrix = (int **)malloc(total_elements * sizeof(int *));
    for (int i = 0; i < total_elements + 1; i++)
        matrix[i] = (int *)malloc(total_elements * sizeof(int));
    for (int i = 0; i < total_elements; i++)
    {
        printf("Enter the value of node %d: ", i + 1);
        scanf("%d", &values_at_nodes[i]);
    }
    printf("\n");
    for (int i = 0; i < total_elements; i++)
    {
        for (int j = 0; j < total_elements; j++)
        {
            if (i == j)
                matrix[i][i] = 0;
            else if (j < i)
                continue;
            else
            {
            retry1:
                printf("Is there an edge between node %d and node %d? (y/n): ", v
alues_at_nodes[i], values_at_nodes[j]);
                scanf(" %c", &choice);
                switch (choice)
                {
                case 'y':
                    matrix[i][j] = 1;
                    matrix[j][i] = 1;
                    break;
                case 'n':
                    matrix[i][j] = 0;
                    matrix[j][i] = 0;
                    break;
                default:
                    printf("Invalid input, enter either y/n\n");
                    goto retry1;
                }
            }
        }
    }
    menu(matrix, values_at_nodes);
}

void menu(int **matrix, int values_at_nodes[])
```

```
{
    system("cls");
    printf("---- MENU ----\n");
    printf("1. B.F.S.\n");
    printf("2. D.F.S.\n");
    printf("3. Exit");
    printf("\n\nEnter your choice: ");
    switch (getch())
    {
    case '1':
        system("cls");
        int start1;
    retry2:
        printf("Enter the starting element: ");
        scanf("%d", &start1);
        for (int i = 0; i < total_elements; i++)
        {
            if (values_at_nodes[i] == start1)
            {
                bfs(matrix, values_at_nodes, i);
                break;
            }
            else if (i + 1 == total_elements)
            {
                printf("Element not found!\n");
                goto retry2;
            }
        }
        printf("\n\nPress any key to go back ");
        if (getch())
            menu(matrix, values_at_nodes);
    case '2':
        system("cls");
        int start2;
    retry3:
        printf("Enter the starting element: ");
        scanf("%d", &start2);
        for (int i = 0; i < total_elements; i++)
        {
            if (values_at_nodes[i] == start2)
            {
                dfs(matrix, values_at_nodes, i);
                break;
            }
            else if (i + 1 == total_elements)
            {
```

```
                printf("Element not found!\n");
                goto retry3;
            }
        }
        printf("\n\nPress any key to go back ");
        if (getch())
            menu(matrix, values_at_nodes);
    case '3':
        system("cls");
        printf("Thank you");
        exit(0);
    default:
        printf("\nIncorrect input, Enter the correct choice");
        menu(matrix, values_at_nodes);
    }
}

void enqueue(int adding_element)
{
    struct node *new = (struct node *)malloc(sizeof(struct node));
    new->data = adding_element;
    new->next = NULL;
    if (front == NULL)
    {
        front = new;
        rear = new;
    }
    else
    {
        rear->next = new;
        rear = new;
    }
}

void push(int adding_element)
{
    struct node *new = (struct node *)malloc(sizeof(struct node));
    new->data = adding_element;
    new->next = front;
    front = new;
}

void pop()
{
    struct node *temp;
    temp = front;
```

```
    front = front->next;
    free(temp);
}

int check_visited(int x, int y, int visited[])
{
    for (int i = 0; i < y; i++)
    {
        if (visited[i] == x)
            return 0;
    }
    return 1;
}

void bfs(int **matrix, int values_at_nodes[], int start_pos)
{
    int visited[total_elements], visited_pos = 0;
    enqueue(values_at_nodes[start_pos]);
    visited[visited_pos] = values_at_nodes[start_pos];
    visited_pos++;
    printf("BFS: ");
    while (front != NULL)
    {
        for (int i = 0; i < total_elements; i++)
        {
            if (values_at_nodes[i] == front->data)
            {
                for (int j = 0; j < total_elements; j++)
                {
                    if ((matrix[i][j] == 1) && check_visited(values_at_nodes[j],
visited_pos, visited))
                    {
                        visited[visited_pos] = values_at_nodes[j];
                        visited_pos++;
                        enqueue(values_at_nodes[j]);
                    }
                }
                printf("%d ", values_at_nodes[i]);
                pop();
                break;
            }
        }
    }
}

void dfs(int **matrix, int values_at_nodes[], int start_pos)
```

```
{
    int visited[total_elements], visited_pos = 0;
    push(values_at_nodes[start_pos]);
    visited[visited_pos] = values_at_nodes[start_pos];
    visited_pos++;
    printf("DFS: ");
    while (front != NULL)
    {
        for (int i = 0; i < total_elements; i++)
        {
            if (values_at_nodes[i] == front->data)
            {
                for (int j = 0; j < total_elements; j++)
                {
                    if ((matrix[i][j] == 1) && check_visited(values_at_nodes[j],
visited_pos, visited))
                    {
                        visited[visited_pos] = values_at_nodes[j];
                        visited_pos++;
                        push(values_at_nodes[j]);
                        break;
                    }
                    else if (j + 1 == total_elements)
                    {
                        printf("%d ", values_at_nodes[i]);
                        pop();
                    }
                }
                break;
            }
        }
    }
}
```

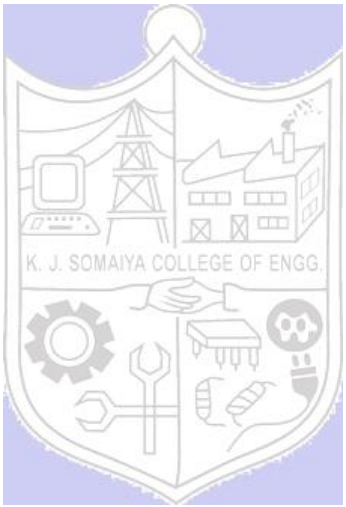
Output:-

```
--------- Enter Graph Values --------------
Enter the no. of nodes: 4
Enter the value of node 1: 1
Enter the value of node 2: 2
Enter the value of node 3: 3
Enter the value of node 4: 4

Is there an edge between node 1 and node 2? (y/n): y
Is there an edge between node 1 and node 3? (y/n): y
Is there an edge between node 1 and node 4? (y/n): n
Is there an edge between node 2 and node 3? (y/n): n
Is there an edge between node 2 and node 4? (y/n): n
Is there an edge between node 3 and node 4? (y/n): y
```

```
---- MENU ----
1. B.F.S.
2. D.F.S.
3. Exit

Enter your choice:
```




```
Enter the starting element: 2
BFS: 2 1 3 4

Press any key to go back
```

```
---- MENU ----
1. B.F.S.
2. D.F.S.
3. Exit

Enter your choice:
```

```
Enter the starting element:
2
DFS: 4 3 1 2

Press any key to go back
```

```
Thank you
```

Outcomes: Apply linear and non linear data structure in application development

---

Conclusion: We implemented a menu driven program in to accept a graph data from the user and traverse it using BFS and DFS technique.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

- Y. Langsam, M. Augenstin and A. Tannenbaum, "Data Structures using C", Pearson Education Asia, 1st Edition, 2002
- E. Balaguruswamy, " Data Structures using C", McGraw Hill Education Private Limited, 2013