



Experiment No. : 6

Title: Graph Traversal using appropriate data structure



Batch: A3**Roll No.:16010423099****Experiment No.: 6**

Aim: Implement a menu driven program to represent a graph and traverse it using BFS technique.

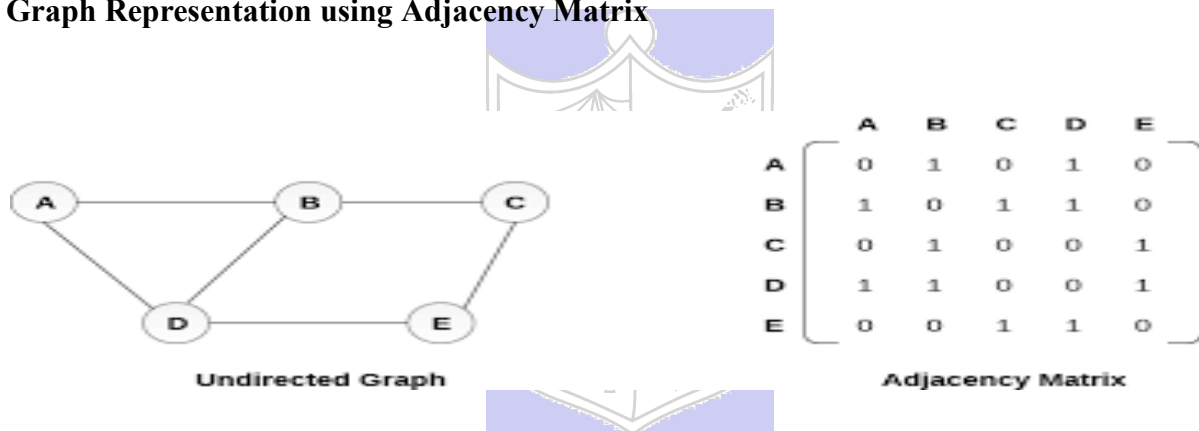
Resources Used: C/ C++ editor and compiler.

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).

Graph Representation using Adjacency Matrix



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 array $visited$ are global and visited is

initialized to false. Initialize, addqueue, emptyqueue, deletequeue are the functions to handle operations on queue.

Algorithm :

Implement the static linear 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

v=deletequeue(q)

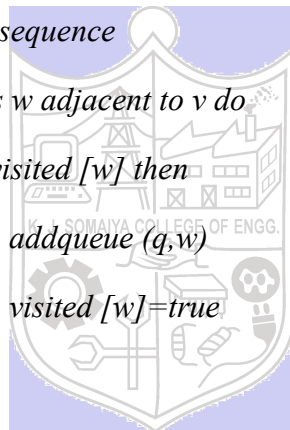
add v into bfs sequence

for all vertices w adjacent to v do

if not visited [w] then

addqueue (q,w)

visited [w]=true



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.

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

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

```
#define MAX_SIZE 10
```

```
typedef struct {
```

```
    int items[MAX_SIZE];
```

```
    int front;
```

```
int rear;
} Queue;

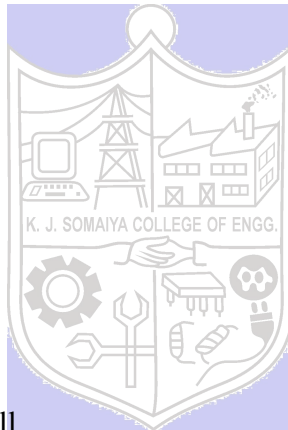
void initQueue(Queue* q) {
    q->front = -1;
    q->rear = -1;
}

int isEmpty(Queue* q) {
    return q->front == -1;
}

int isFull(Queue* q) {
    return q->rear == MAX_SIZE - 1;
}

void enqueue(Queue* q, int value) {
    if (isFull(q)) return; // Ignore if full
    if (q->front == -1) q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
}

int dequeue(Queue* q) {
    if (isEmpty(q)) return -1; // No operation if empty
    int item = q->items[q->front];
    if (q->front >= q->rear) {
        q->front = -1;
    }
}
```



```

    q->rear = -1;
} else {
    q->front++;
}
return item;
}

```

```

void bfs(int graph[MAX_SIZE][MAX_SIZE], int startRow, int numRows) {

```

```

    Queue q;

```

```

    initQueue(&q);

```

```

    int visited[MAX_SIZE] = {0};

```

```

    visited[startRow] = 1;

```

```

    enqueue(&q, startRow);

```

```

    printf("BFS Sequence: ");

```

```

    while (!isEmpty(&q)) {

```

```

        int v = dequeue(&q);

```

```

        printf("%d ", v);

```

```

    for (int w = 0; w < numRows; w++) {

```

```

        if (graph[v][w] == 1 && !visited[w]) {

```

```

            enqueue(&q, w);

```

```

            visited[w] = 1;

```

```

        }

```

```

    }

```

```

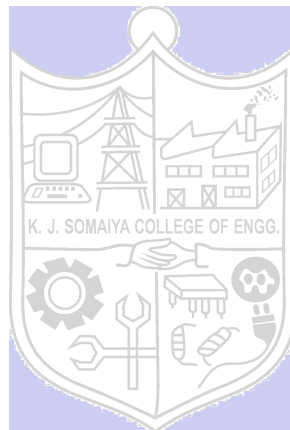
}

```

```

printf("\n");

```

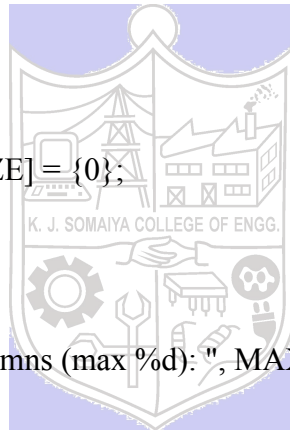


```
}
```

```
void displayGraph(int graph[MAX_SIZE][MAX_SIZE], int numRows) {
    printf("Adjacency Matrix:\n");
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numRows; j++) {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }
}
```

```
int main() {
    int graph[MAX_SIZE][MAX_SIZE] = {0};
    int numRows;

    printf("Enter number of rows/columns (max %d): ", MAX_SIZE);
    scanf("%d", &numRows);
```



```
if (numRows > MAX_SIZE) {
    printf("Number exceeds maximum limit.\n");
    return 1;
}

printf("Enter the adjacency matrix (%d x %d):\n", numRows, numRows);
for (int i = 0; i < numRows; i++) {
    for (int j = 0; j < numRows; j++) {
        scanf("%d", &graph[i][j]);
```

```
    }  
}  
  
printf("\n");  
displayGraph(graph, numRows);  
  
int startRow;  
printf("Enter starting row for BFS: ");  
scanf("%d", &startRow);  
  
bfs(graph, startRow, numRows);  
  
return 0;  
}
```



```
D:\MinGW\stuff\16010423099_EXP6_DS.exe  
Enter number of rows/columns (max 10): 3  
Enter the adjacency matrix (3 x 3):  
0 1 0  
1 1 0  
0 0 0  
  
Adjacency Matrix:  
0 1 0  
1 1 0  
0 0 0  
Enter starting row for BFS: 0  
BFS Sequence: 0 1  
  
Process returned 0 (0x0)   execution time : 6.870 s  
Press any key to continue.
```

Outcomes:

CO2: Apply linear and non-linear data structure in application development.

Conclusion:


Program executed successfully and applied the BFS algorithm using an adjacency matrix to traverse the graph, correctly reflecting reachable nodes from the starting point.

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.
 - Vlab on BFS

