Breadth First Search

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 40
struct queue {
int items[SIZE];
int front;
int rear;
};

struct queue* createQueue(); void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);
struct node {
int vertex;
struct node* next;
};
struct node* createNode(int);

struct Graph {
int numVertices;
struct node** adjLists;
int* visited;
};
// BFS algorithm
void bfs(struct Graph* graph, int startVertex) {
struct queue* q = createQueue();

graph->visited[startVertex] = 1;
enqueue(q, startVertex);
```

```
while (!isEmpty(q)) {
printQueue(q);
int currentVertex = dequeue(q);
printf("Visited %d\n", currentVertex);

struct node* temp = graph->adjLists[currentVertex];
while (temp) {
int adjVertex = temp->vertex;

if (graph->visited[adjVertex] == 0) {
graph->visited[adjVertex] = 1;
enqueue(q, adjVertex);
}
temp = temp->next;
}
}
}

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



// Creating a 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;
}

// Create a queue
struct queue* createQueue() { struct queue* q = malloc(sizeof(struct queue));
q->front = -1;
q->rear = -1;
return q;
}

// Check if the queue is empty int isEmpty(struct queue* q) {
if (q->rear == -1)
```

```
return 1;
else
return 0;


// Adding elements into queue
void enqueue(struct queue* q, int value) {
if (q->rear == SIZE - 1)
printf("\nQueue is Full!!");
else {
if (q->front == -1)
q->front = 0;
q->rear++;
q->items[q->rear] = value;
}


// Removing elements from queue
int dequeue(struct queue* q) {
int item;
if (isEmpty(q)) {
printf("Queue is empty"); item = -1;
} else {
item = q->items[q->front];
q->front++;
if (q->front > q->rear) {
printf("Resetting queue ");
q->front = q->rear = -1;
}
}
return item;
```

```
// Print the queue
void printQueue(struct queue* q) {
int i = q->front;

if (isEmpty(q)) {
printf("Queue is empty");
} else {
printf("\nQueue contains \n");
for (i = q->front; i < q->rear + 1; i++) { printf("%d ", q->items[i]);
}
}
int main() {
struct Graph* graph = createGraph(6);
addEdge(graph, 0, 1);
addEdge(graph, 0, 2); addEdge(graph, 1, 2);
addEdge(graph, 1, 4);
addEdge(graph, 1, 3);
addEdge(graph, 2, 4);
addEdge(graph, 3, 4);
bfs(graph, 0);
return 0;
```

Output:

1.Insert Beg

2.Insert Middle

3.Insert End

4.Delete Beg

5.Delete Middle

6.Delete End

7.Find

8.Traverse

9.Exit

Enter your choice : 1

Enter the element : 40

Enter your choice : 1

Enter the element : 30

Enter your choice : 1

Enter the element : 20

Enter your choice : 1

Enter the element : 10

Enter your choice : 8 10 20 30 40

Enter your choice : 7

Enter the element : 30

Element found...!

Enter your choice : 1

Enter the element : 5

Enter your choice : 8 5 10 20 30 40

Enter your choice : 3

Enter the element : 45

Enter your choice : 9