Week-9
Breadth First Search

26/02

```
#include<stdio.h>
#include<stdlib.h>
#define ~~MAX~~SIZE 100
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;
  };
void bfs(struct Graph* graph, int startVertex);
struct Graph* createGraph(int vertices);
void addEdge(struct Graph * graph, int src, int dest);
struct queue* createQueue();
int isEmpty(struct queue* q);
void enqueue(struct queue *q, int value);
int dequeue (struct queue *q);
void printQueue(struct queue *q);
```

```
struct node * createNode(int v) {
 struct node * newNode = malloc(sizeof(struct node));
 newNode -> vertex = v;
 newNode -> next = NULL;
 return newNode;
}
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;
 }
void addEdge(struct Graph * graph, int src, int dest) {
  struct node * newNode = createNode(dest);
  newNode -> next = graph -> adjLists[src];
  graph -> adjLists[src] = newNode;
  newNode = createNode(src);
  newNode -> next = graph -> adjLists[dest];
  graph -> adjLists[dest] = newNode;
 }
                26/2/24
struct queue * createQueue() {
   struct queue * q = malloc(sizeof(struct queue));
   q -> front = -1;
   q -> rear = -1;
   return q;
 }
int isEmpty(struct queue *q) {
```

```
if(q->rear==-1)
   return1;
else
   return0;
}
void enqueue(struct queue *q, int value){
     if(q->rear==SIZE-1)
       printf("Queue is Full!!");
     else{
         if(q->front==-1)
         q->front=0;
         q->rear++;
         q->items[q->rear]=value;
     }
  }
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;
 }
void printQueue(struct queue *q){
    int i=q->front;
   if(isEmpty(q)){
        printf("Queue is empty"); }
   else{ printf("\n Queue contains \n");
   for (i=q->front; i<q->rear+1; i++){
```

```
printf("%d", q->items[i]);
}}
 }
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->adjList[currentVertex];

    while (temp) {
           int adjvertex = temp->vertex;
     if(graph->visited[adjVertex]==0) {
        graph->visited[adjVertex]=1;
         enqueue(q, adjVertex);
     }
    temp = temp->next;
    }
  }
}
int main() {
  int numVertices, numEdges;
  printf("Enter no of vertices:");
  scanf("%d", &numVertices);
 struct Graph *graph = createGraph(numVertices);
  printf("Enter no of Edges:");
  scanf("%d", &numEdges);
 for (int i=0; i<numEdges; i++) {
     int src, dest;
   printf("Enter edge %d (source destination)?", i+1);
   scanf("%d%d", &src, &dest);
```

```
    [illegible]Edge(graph, src, dest);
  }
  int startVertex;
printf("Enter starting vertex for BFS: ");
 scanf("%d", &startVertex);
 bfs(graph, startVertex);
return 0;
}
```

O/P:

```
Enter edge 1 (source destination): 1 2
Enter edge 2 (source destination): 1 3
Enter edge 3 (source destination): 1 4
Enter edge 4 (source destination): [illegible]
Enter edge 5 (source destination): [illegible]
Enter edge 6 (source destination): [illegible]
Enter edge 7 (source destination): [illegible]
Enter starting node for BFS: [illegible]
BFS traversal starting [illegible]
```