26/2/24

1> BFS

→ #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);
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 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

while (temp) {

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

DOMS Page No. Date / /

```
struct node * createNode(int v) {
    struct node * newnode;
newnode = (struct node *) malloc (sizeof(
newnode -> vertex = v
newnode -> next = NULL;
    return newnode;
}

struct node * newNode,
 newnode = createNode (dest);
newnode -> next = graph -> adjLists[src];
graph -> adjLists [src] = newnode;

newNode = createNode [src];
newNode -> next = graph -> adjLists [dest];
graph -> adj [dest] = newnode; }
```

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

int isEmpty(struct queue *q){
if (q->rear == -1){
return 1;
else
return 0;
}

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->item[q->rear] = value;
  }
}

void printqueue(struct queue *q){
  int i = q->front;
  if(isEmpty(q)){
    printf("Queue is Empty");
  }else{
    printf("Queue Contains");
    for(int i = q->front; i<q->rear+1;i++)
    printf("%d", q->items[i]);
  }
}
```

```
int main() {
    struct Graph* graph = createGraph(5);
    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;
}
```

O/P:

Queue Contain

0 resetting queue visited 0

Queue contains

2 1 visited 2.

Queue contains
1 4 visited 1

Queue contains
4 3 visited 4

Queue contains
3 Resetting queue visited 3

DOMS Page No. Date / /

DFS

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

void DFS (struct Graph *graph, int vertex)
{
    struct node *adjlist = graph->adjlist[vertex];
    struct node *temp = adjlist;

    graph->visited[vertex] = 1;
    printf("Visited %d", vertex);

    while (temp != NULL) {
        int connectedVertex = temp->vertex;
        if (graph->visited[connectedVertex] == 0) {
            DFS(graph, connectedVertex);
        }
        temp = temp->next;
    }
}
```

```
void printGraph(struct Graph* graph)
    int v;
for (v=0; v< graph->numVertices; v++){
    struct node* temp = graph->adjLists[v];
printf("Adjacency list of vertex %d\n", v);
    while (temp){
printf("%d -> ", temp->vertex);
    temp = temp->next;
} printf("\n"); }
int main(){
struct Graph* graph(4);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 2, 3);
print Graph(graph);
    DFS(graph, 2);
    return 0;
}.
```

O/P :

Adjacency List of vertex 0
2 → 1

Adjacency list of vertex 1
2 → 0

Adjacency list of vertex 2
3 → 1 → 0

Adjacency list of vertex 3
2

Visited 2
visited 3
visited 1
visited 0

4/3/24