26/2/24

Week - 9

BFS ⇒ 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;
  };
```

```
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;
    }
  }
}

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;
}

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)
        return 1;
    else
        return 0;
}
```

```
void enqueue(struct queue *q, int value){
    if (q->rear == SIZE-1)
        printf("\n 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)){
        pf("Queue is Empty");
        item = -1;
    }else{
        item = q->items[q->front];
        q->front++;
        if (q->front > q->rear){
            pf("Resetting queue");
            q->front = q->rear = -1;
        }
    }
    return item;
}
```

```
void printQueue (struct queue *q){
    int i = q->front;
if (isEmpty(q)){
    pf("Queue is empty");

}else{
    pf("\n Queue contains \n");
    for (i = q->front ; i<q->rear +1; i++){
        pf(" %d", q->items[i]);
    }
}
}
int main(){
    struct graph * graph = create Graph(6);
add Edge (graph, 0,1);
add Edge (graph, 0,2);
addEdge (graph,1,2);
addEdge (graph,1,3);
addEdge (graph, 2,4);
addEdge (graph, 3.4);

bfs (graph,0);
    return 0;
}
```

Output

Queue Contains
0 resetting queue visite 0.

Queue contains
2 1 visited 2

Queue contains
1 4 visitied 1

Queue contains.
4 3 visited 4

queue contains
3 resetting queue visited 3.

② DFS

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int vertex;
    struct node *next;
};
struct node *createNode(int v);
struct Graph {
    int numVertices;
    int *visited;
    struct node **adjLists;
};
void DFS(struct Graph *graph, int vertex)
{
    struct node *adjList = graph->adjLists[vertex];
    struct node *temp = adjList;
    graph->visited[vertex]=1;
    printf("Visited %d \n", 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("\n Adjacency List of vertex %d \n ", v);
        while(temp){
            printf("%d -> ", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}
int main(){
    struct Graph* graph = createGraph(4);
    addEdge(graph,0,1);
    addEdge(graph,0,2);
    addEdge(graph,1,2);
    addEdge(graph,2,3);
    printGraph(graph);
    DFS(graph,2);
    return 0;
}
```

Output : 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