LAB-8

1. Breadth First Search

=>

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

#define MAX_NODES 100

struct Node {
    int data;
    struct Node* next;
};

struct Graph {
    int numNodes;
    struct Node* adjLists[MAX_NODES];
    int visited[MAX_NODES];
};

struct Node* createNode(int data) {
    struct Node* newNode = malloc(sizeof
                                  (struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

26/2/24

```
struct Graph* createGraph(int n) {
    struct Graph* graph = malloc(sizeof
                          (struct Graph
    graph->newNode = n;
    for (int i=0; i<n; 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] = newNodes
}
```

~~void BFS(~~

```
void BFS (struct Graph* graph, int startNode){
    int queue[MAX_NODES]
    int front=0 , rear=0;

    graph->visited[startNode]=1;
    queue[rear++]=startNode;

    while(front<rear){
        int current = queue[front++];
        printf("%d ", current);

        struct Node* temp =graph->
                adjlists[current];
        while(temp){
            int adjNode =temp->data;
            if(!graph->visted[adjNode])
            {
                graph->visited[adjNode]
                =1;

                queue[rear++]=adjNode;
            }
            temp= temp->next;
        }
    }
}
```

```
int main(){
    int newNodes;
    printf("Enter the number of nodes: ");
    scanf("%d", &newNodes);

    struct Graph *graph = createGraph(new

    int newEdges;
    printf("Enter the number 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
        addEdges(graph, src, dest);
    }

    int startNode;
    printf("Enter the starting node fo
            BFS traversal: ");
    scanf("%d", &startNode);

    printf("BFS traversal starting from
            node %d: ", startNode);
    BFS(graph startNode);
    return 0
}
```

```
Enter the number of nodes: 5
Enter the number of edges: 4
Enter edge 1 (source destination): 0
1
Enter edge 2 (source destination): 0
2
Enter edge 3 (source destination): 2
3
Enter edge 4 (source destination): 1
4
Enter the starting node for BFS traversal: 0
BFS traversal starting from node 0: 0 2 1 3 4
```

2. Depth first Search

=>

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

#define MAX_NODES 100

struct Node{
        int data;
        struct Node* next;
};

struct Graph{
        int numNodes;
        struct Node * adjLists[MAX_NODES];
        int visited[MAX_NODES];
};

struct Node* createNode(int data){
        struct Node* newNode = malloc(size
                                (struct Node));
        newNode -> data = data;
        newNode -> next = NULL;
        return newNode;
}
```

```
struct Graph *createGraph(int n) {
    struct Graph* graph = malloc(sizeof
                          (struct Node));
    graph->newNode = n;
    for(int i=0; i<n; 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;
}

void DFS(struct Graph* graph, int startNode){
    graph->visited[startNode] = 1;
    printf("%d", startNode);

    struct Node *temp = graph->adjLists
                        [startNode];
```

```
        while (temp) {
            int adjNode = temp->data;
            if(!graph->visited[adjNode])
            {
                DFS(graph, adjNode);
            }
            temp = temp->next;
        }
}

int main() {
    int numNodes;
    printf("Enter the number of node:");
    scanf("%d", &newNodes);

    struct Graph* graph = createGraph(newNode

    int numEdges;
    printf("Enter ther number of edge:");
    scanf("%d", &newEdges);

    for (int i=0; i<numEdges; i++){
        int src, dest;
        printf("Enter edge %d (source
                destination): ", i+1);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }
```

```
    int startNode;
    printf("Enter the starting node for
            DFS traversal: ");
    scanf("%d", &startNode);
    printf("DFS traversal starting for node
            %d: ", startNode);
    DFS(graph, startNode);
    return 0;
}
```

Output:-

```
Enter the number of nodes: 5
Enter the number of edges: 4
Enter edge 1 (source destination): 0 1
Enter edge 2 (source destination): 0 2
Enter edge 3 (source destination): 1 3
Enter edge 4 (source destination): 1 4

Enter the starting node for DFS
traversal: 0

DFS traversal starting from node 0:
 0 1 3 4 2.
```

26/2/24

```
Enter the number of nodes: 5
Enter the number of edges: 4
Enter edge 1 (source destination): 0
1
Enter edge 2 (source destination): 0
2
Enter edge 3 (source destination): 1
3
Enter edge 4 (source destination): 1
4
Enter the starting node for DFS traversal: 0
DFS traversal starting from node 0: 0 2 1 4 3
```

* leetcode :-
Delete Node in a BST

=>

```
/**
struct TreeNode* Smallest (struct TreeN
                           root)
{
    struct TreeNode * cur = root;
    while (cur -> left != NULL)
    {
        cur = cur -> left;
    }
    return cur;
}

struct TreeNode* deleteNode (struct
    TreeNode* root, int Key)
{
    if (root == NULL)
    {
        return root;
    }

    if(Key < root -> val)
    {
        root -> left = delete (root -> left, Key
    }
```

```
else if (key > root->val
{
    root->right = deleteNode(root->right,
                             key);
}
else
{
    if (root->left == NULL)
    {
        struct TreeNode *temp = root->
                                right;
        free(root);
        return temp;
    }
    else if (root->right == NULL)
    {
        struct TreeNode *temp = root->
                                left;
        free (root);
        return temp;
    }

    struct TreeNode *temp = Smallest(root->
                                     right)
    root->val = temp->val;
    root->right = deleteNode(root->right,
      root->val);
}
return root;
}
```

- Test_Result.

Input

root =

[5,3,6,2,4,null,7]

Key = 3

Output

[5,4,6,2,null,null,7]

Expected

[5,4,6,2,null,null,7]

26/12/24

* Leetcode :-
Find the Bottom leftvalue ;

⇒

```
int findBottomLeftValue (struct TreeNode* root)
{
    Struct TreeNode * queue [10000];
    int front = 0, rear = 0, levelSize=0
    leftmostValue =0;

    if (root == NULL){
        return 0;
    }

    queue [rear++] = root;

    while (front < rear){
        levelSize = rear - front;

        for (int i=0; i< levelSize; i++){
            Struct TreeNode* currentNode
                = queue [front++];

            if (i==0) {
                leftmostValue = currentNode
                    ->val;
            }
```

```
if(currentNode->left != NULL){
    queue[rear++] = currentNode->left;
}

if(currentNode->right != NULL){
    queue[rear++] = currentNode->
                    right;
    }
  }
}
return leftmostValue;
}
```

Test_results

Input =
root = [1, 2, 3, 4, null, 5, 6, null, null, 7]

~~Out~~

Output
7

Expected
7.