Lab ⇒ 26/02/2024

SURYA Gold
Date _____ Page _____

Q.1.) Implement DFS traversals.

Ans ⇒

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 20
int graph[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES]; int n;
void dfs(int start){
    int i;
    visited[start]=1;
    printf("visited %d\n", start);
    for(i=0; i<n; i++){
        if(graph[start][i] && !visited[i]){
            dfs[i];
        }
    }
}
int main(){
    int i, j; int start;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix:\n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            scanf("%d", &graph[i][j]);
        }
    }
    printf("Enter the starting vertex for
            DFS: ");
    scanf("%d", &start);
```

```
for ( i=0; i<n; i++) {
        visited [i] = 0;
}
dfs (start);
return 0;
}
```

Output -:

```
Enter the number of vertices: 4
Enter the adjacency matrix:
1 0 0 1
0 1 0 0
1 1 1 0
1 0 0 0

Entering the starting vertex for DFS : 0
visited 0
visited 3
```

Q.20) Implement BFS.

Ans ⇒

```
#include <stdio.h>
void bfs (int a[10][10], int n, int u){
        int f, r, q[10], v;
        int s[10] = {0};
    printf("The nodes visited from %d", u);
    f=0;
    r=-1;
    q[++r] = u;
    s[u] = 1;
    printf("%d", u);
    while( f<=r) {
```

```
        u = q[f++];
        for (v=0; v<n; v++) {
            if ((a[u][v]==1) && (s[v]==0))

            printf ("%d", v);
            s[v] = 1;
            q[++r] = v;
        }
    }
    printf ("\n");
}

void main () {
    int n, a[10][10], source, i, j;
    printf ("\n Enter no. of nodes : ");
    scanf ("%d", &n);
    printf ("\n Enter the adjacency matrix :\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            scanf ("%d", &a[i][j]);
        }
    }
    for (source=0; source<n; source++) {
        bfs (a, n, source);
    }
}
```

Output :-

Enter no. of nodes : 3

Enter the adjacency matrix:

1 1 1
1 0 0
0 1 0

The node visited from 0 0 1 2
The node visited from 1 1 0 2
The node visited from 2 2 1 0

Q.) Delete a node in BST ⇒ Leetcode.

Ans ⇒

```
struct TreeNode* deleteNode(struct TreeNode* root, int key){

if (root == NULL)
 return root;

if (key < root->val){
    root->left = deleteNode(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 = minValueNode
(root->right);
    root->val = temp->val;
    root->right = deleteNode(root->right,
                             temp->val);
}
return root;
}

void inOrder(struct TreeNode* root) {
    if (root != NULL) {
        inOrder(root->left);
        printf("%d", root->val);
        inOrder(root->right);
    }
}

struct TreeNode* minValueNode(struct
    TreeNode* node) {
        struct TreeNode* current = node;
    while(current && current->left != NULL)
    {
        current = current->left;
    }
    return current;
}
```

Q:) Bottom left tree value → Leetcode

Ans →)

```
int findBottomLeftValue(struct TreeNode* root) {
struct TreeNode* queue[1007];
int front = 0, rear = 0;
queue[rear++] = root;
int leftmostValue = root->val;
while (front < rear) {
    int levelSize = rear - front;
  for(int i=0; i< levelSize; i++) {
    struct TreeNode* current = queue[front++];

if (i == 0)
{
    leftmostValue = current->val;
}
else if (current->left != NULL) {
    queue[rear++] = current->left;
}

else if (current->right != NULL) {
    queue[rear++] = current->right;
}
}

return leftmostValue;

}
```