

```

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 BFS (struct graph * graph, int startNode) {
    int queue [MAXNODES];
    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->visited [adjNode]) {
                graph->visited [adjNode] = 1;
                queue [rear++] = adjNode;
            }
            temp = temp->next;
        }
    }
}

```

A tree diagram with root node A. A is connected to B and C. B is connected to D and E. C is connected to F. The nodes are labeled with numbers: A=26, B=12, C=24, D=7, E=7, F=7. A red line connects A to D.



① BFS

#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;

#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 = (struct Node\*) malloc

(sizeof (struct Node));

newNode-&gt;data = data;

newNode-&gt;next = NULL;

return newNode; }

struct Graph\* createGraph (int n) {

struct Graph\* graph = (struct Graph\*) malloc

(sizeof (struct Graph));

graph-&gt;numNodes = n;

for (int i = 0; i &lt; n; i++) {

graph-&gt;adjLists[i] = NULL;

graph-&gt;visited[i] = 0;

}

return graph;

}



```

int main() {
    int numNodes;
    printf("Enter the no. of nodes: ");
    scanf("%d", &numNodes);
    struct graph * graph = createGraph(numNodes);
    int numEdges;
    printf("Enter the no. of edges: ");
    scanf("%d", &numEdges);
    for(int i=0; i<numEdges; i++) {
        int src, dest;
        printf("Enter edge %d (source destination):", i);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }
    int startNode;
    printf("Enter the starting node for BFS traversal:");
    scanf("%d", &startNode);
    printf("BFS traversal starting from node %d", startNode);
    BFS(graph, startNode);
    return 0;
}

```

o/p) Enter the no. of nodes: 5  
 Enter the no. of edges: 4  
 Enter edge 1: 0 1  
 Enter edge 2: 0 2  
 Enter edge 3: 1 3  
 Enter edge 4: 1 4  
 Enter edge starting node for BFS traversal: 0  
 BFS traversal starting from node: 0 2 1 4 3



```

void addEdge (struct Graph* graph, int src, int dest) {
    struct Node* newNNode = createNode (dest);
    newNNode->next = graph->adjList[src];
    graph->adjList[src] = newNNode;
    newNNode = createNode (src);
    newNNode->next = graph->adjList[dest];
    graph->adjList[dest] = newNNode;
}

```

```

void DFS (struct Graph* graph, int startNode) {
    graph->visited[startNode] = 1;
    printf ("%d", startNode);
    struct Node* temp = graph->adjList[startNode];
    while (temp) {
        int adjNode = temp->data;
        if (!graph->visited[adjNode]) {
            DFS (graph, adjNode);
        }
        temp = temp->next;
    }
}

```

```

int main () {
    int numNodes;
    printf ("Enter the no. of nodes: ");
    scanf ("%d", &numNodes);
    struct Graph* graph = createGraph (numNodes);
    int numEdges;
    printf ("Enter the no. of edges: ");
    scanf ("%d", &numEdges);
    for (int i=0; i<numEdges; i++) {
        int src, dest;
        printf ("Enter edge %d (source destination): ", i+1);
    }
}

```



DFS  
 #include <stdio.h>  
 #include <stdlib.h>  
 #define MAX\_NODES 100

struct node {  
 int data;  
 struct node\* next; };

struct graph {  
 int numNodes;  
 struct node\* adjList [MAX\_NODES];  
 int visited [MAX\_NODES]; };

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

struct graph\* createGraph (int n) {  
 struct graph\* g = (struct graph\*) malloc  
 (sizeof (struct graph));  
 g->numNodes = n;  
 for (int i = 0; i < n; i++) {  
 g->adjList[i] = NULL;  
 g->visited[i] = 0; }  
 return g;  
}



string ("i.d.i.d", src, dest);  
 addEdge (graph, src, dest);

int startNode;

printf ("Enter the starting node for DFS traversal: ");  
 scanf ("%d", &startNode);

printf ("DFS traversal starting from node i.d: ", startNode);  
 DFS (graph, startNode);  
 return 0;

}

→ Enter the no. of nodes : 5  
 Enter the no. of edges : 4  
 Enter edge 1 : 0 1  
 Enter edge 2 : 0 2  
 Enter edge 3 : 1 3  
 Enter edge 4 : 4 2  
 Enter the starting node for DFS traversal : 0  
 DFS traversal from node 0 : 0 1 3 4 2

Ques 1) Delete node in a BST

struct Treenode \* minValueNode (struct Treenode \* node)

{ struct Treenode \* current = node;  
 while (current->left != NULL)

current = current->left;

return current;

struct Treenode \* deleteNode (struct Treenode \* root, int key)

{ if (root == NULL)

return root;

if (key < root->val)
 { root->left = deleteNode (root->left, key);



```

    bfsQueue[set] = pCur->left;
    set += 1; }

    if (pCur->right != NULL) {
        bfsQueue[set] = pCur->right;
        set += 1; } } while (get < set);

    return p->valInRow;
}

```



```

else if (val > root->val)
    root->right = deleteNode (root->right, val);
else if (val < root->val)
    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;
}

```

2) Find bottom left tree value.

```

typedef struct TreeNode TreeNode;
#define MAX_NODE (10000)
int findBottomLeftValue (const TreeNode* root) {
    assert (root != NULL);
    int firstValInRow;
    const TreeNode* bfsQueue [MAX_NODE];
    int get = 0, set = 0;
    bfsQueue [set] = root;
    set += 1;
    do {
        firstValInRow = bfsQueue [get] -> val;
        for (int rest = set - get; rest > 0; rest -= 1) {
            const TreeNode* cur = bfsQueue [rest];
            get++;
            if (cur->left != NULL)

```