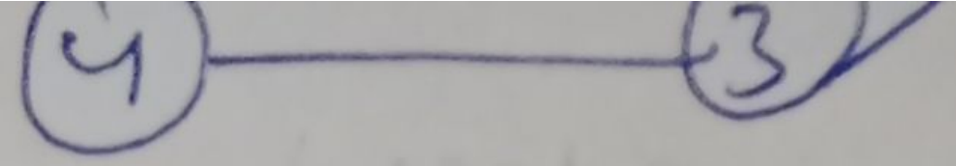


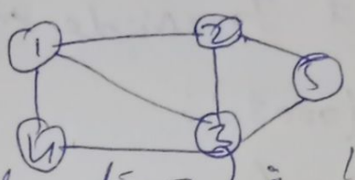
(2) DFS

```
#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* createNode(int v);
struct Graph* createGraph(int vertices);
void addEdge(struct Graph* graph, int src, int dest);
void printGraph(struct Graph* graph);
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;
    }
}
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;
}
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() {
    int numVertices, numEdges;
    printf("Enter the number of vertices");
    scanf("%d", &numVertices);
    struct Graph* graph = createGraph(numVertices);
    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);
        scanf("%d %d", &src, &dest);
```

```
    addEdge(graph, src, dest);
}
printGraph(graph);
int startVertex;
printf("Enter the starting vertex for DFS: ");
scanf("%d", &startVertex);
DFS(graph, startVertex);
return 0;
}
```



Output:

```
Enter edge 1 (source destination): 1 2
Enter edge 2 (source destination): 1 3
Enter edge 3 (source destination): 1 4
Enter edge 4 (source destination): 2 3
Enter edge 5 (source destination): 2 5
Enter edge 6 (source destination): 3 4
Enter edge 7 (source destination): 3 5
Enter starting node for DFS traversal: 1
DFS traversal starting from node 1: 1 4 3 5 2
```

(3) Delete Node in a BST

```
struct TreeNode* smallest(struct TreeNode* 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; }
```

26/2/24

```
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 = smallest(root->right);
    root->val = temp->val;
    root->right = deleteNode(root->right, root->val);
}
return root;
}
```

⑧ Find Bottom Left Tree Value

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