```cpp
// incomplete for Deletion on head
#include<iostream>
#include<queue>
using namespace std;

class edge{
public:
    char dest;
    edge* link;
    edge(char ch){
        dest=ch;
        link=NULL;
    }
};
class node{
public:
    char info;
    node* next;
    edge* adj;
    node(char val){
        info=val;
        next=NULL;
        adj=NULL;
    }
};
```

```cpp
//insert node At last
void insertNode(node* &start,char ele){
        node* n=new node(ele);
        if(start==NULL){
            start=n;
            return;
        }
        node*temp=start;
        while(temp->next!=NULL){
            temp=temp->next;
        }
        temp->next=n;
        return;
}
// Find Node
node* findNode(node* start,char ch){
        node* temp=start;
        if(start==NULL){
            return NULL;
        }
        while(temp!=NULL){
            if(temp->info==ch){
                return temp;
```

```cpp
                }
            temp=temp->next;
        }
        return NULL;
}
//Create Egde between two nodes
void createEdge(node* &start,char ch1,char ch2){
    node* temp=start;
    node* n1=findNode(temp,ch1);
    node* n2=findNode(temp,ch2);
    if(n1!=NULL and n2!=NULL){
        edge* ed=new edge(ch2);
        if(n1->adj==NULL){
            n1->adj=ed;
            return;
        }
        edge* temp_edge=n1->adj;
        if(temp_edge->link==NULL){
            temp_edge->link=ed;
            return;
        }
        while(temp_edge->link!=NULL){
            temp_edge=temp_edge->link;
        }
        temp_edge->link=ed;
        return;
    }
    else{
        cout<<"Edge is not possible"<<endl;
        return;
    }
}
//Delete edge
void DeleteEdge(node* start,char ch1,char ch2){
    node* n1=findNode(start,ch1);
    if(n1==NULL){
        return;
    }
    if(n1->adj==NULL){
        return;
    }
    edge* temp_E=n1->adj;
    if(temp_E->dest == ch2){
        // edge* todele=temp_E;
        n1->adj=temp_E->link;
        delete temp_E;
        return;
    }
```

```cpp
        while(temp_E->link!=NULL and temp_E->link->dest!=ch2){
            temp_E=temp_E->link;
            // cout<<"*";
        }
        edge* todelete=temp_E->link;
        if(temp_E->link!=NULL){
        temp_E->link=temp_E->link->link;
        }
        delete todelete;
}
node* DeleteNode(node* start,char val){
    if(start==NULL){
        return NULL;
    }
    // todelete node from adj list of vertices
    node* tp=start;
    while(tp!=NULL){
        DeleteEdge(start,tp->info,val);
        tp=tp->next;
    }
    // todelete adj list of Delnode
    node* tp2=findNode(start,val);
    edge* del=tp2->adj;
    while(del!=NULL){
        char current_edge_info = del->dest;
        del=del->link;
        DeleteEdge(start,val,current_edge_info);
    }
    if(start->info == val){
        node* new_start = start->next;
        delete start;
        return new_start;
    }
    node* temp=start;
    while(temp->next != NULL){
        if(temp->next->info==val){
            break;
        }
        temp=temp->next;
    }
    node* todelete=temp->next;
    if(todelete == NULL){
        cout<<"Element doesn't exist in graph";
        return start;
    }
    temp->next=temp->next->next;
    delete todelete;
    return start;
```

```cpp
}

//print node
void printNode(node* temp){
    if(temp==NULL){
        cout<<"EMPTY LIST"<<endl;
        return;
    }
    while(temp!=NULL){
        cout<<temp->info<<"->";
        temp=temp->next;
    }
    cout<<"NULL"<<endl;
}
// print adjacency list
void printAdj(node* start,char ch){
    node* temp=findNode(start,ch);
    if(temp==NULL){
        cout<<"EMPTY LIST"<<endl;
        return;
    }
    cout<<temp->info<<"-> ";
    edge* temp_E=temp->adj;
    while(temp_E!=NULL){
        cout<<temp_E->dest<<",";
        temp_E=temp_E->link;
    }
    cout<<"NULL"<<endl;
}
// calculate indegree
void Cal_degree(node* start,char ch){
    node* temp=findNode(start,ch);
    if(temp==NULL){
        cout<<"EMPTY LIST"<<endl;
        return;
    }
    int count=0;
    cout<<"Degree of "<<temp->info<<"-> ";
    edge* temp_E=temp->adj;
    while(temp_E!=NULL){
        count++;
        temp_E=temp_E->link;
    }
    cout<<" "<<count<<endl;
}
int cal_indegree(node* start,char ch){    // not good approach
    node* temp1=findNode(start,ch);
    if(temp1==NULL){
        cout<<"NODE IS NOT PRESENT IN LIST"<<endl;
```

```cpp
            return -1;
        }
        // cout<<"#"<<endl;
        node* temp=start;
        int count=0;
        while(temp!=NULL){
            edge* temp_E=temp->adj;
            while(temp_E!=NULL){
                if(temp_E->dest==ch){
                    count++;
                }
                temp_E=temp_E->link;
            }
            temp=temp->next;
        }
        return count;
}
bool checkEdge(node* start,char ch1,char ch2){
        node* temp=findNode(start,ch1);
        edge* temp_E=temp->adj;
        while(temp_E!=NULL){
            if(temp_E->dest==ch2){
                return true;
            }
            temp_E=temp_E->link;
        }
        return false;
}
//Topological Sort
void topologicalSort(node* start){
        queue<char> q1;
        node* temp=start;
        while(temp!=NULL){
            if(cal_indegree(start,temp->info)==0){
                q1.push(temp->info);
                cout<<"temp-info: "<<temp->info<<endl;
            }
            temp=temp->next;
        }
        while(!q1.empty()){
            char c=q1.front();
            q1.pop();
            node* dummy=findNode(start,c);
            edge* temp_E=dummy->adj;
            cout<<c<<" ";
            while(temp_E!=NULL){
                char ch=temp_E->dest;
                if(cal_indegree(start,ch)==1 and checkEdge(start,c,ch)){
```

```cpp
                    q1.push(ch);
                }
                temp_E=temp_E->link;
            }
        start=DeleteNode(start,c);
        }
    cout<<endl;
}
```

```cpp
int main(){
    char c,ch;
    int n,choice;
    node* start=NULL;
    cout<<"Enter the number of vertices: ";
    cin>>n;
    while(n){
        cout<<"Enter vertex: ";
        cin>>c;
        insertNode(start,c);
        n--;
    }
    int edges;
    cout<<"Enter the number of edges: ";
    cin>>edges;
    while(edges){
        char ch1,ch2;
        cout<<"Enter Vertices: "<<endl;
        cin>>ch1>>ch2;
        createEdge(start,ch1,ch2);
        edges--;
    }
    cout<<"ALL THE EDGES ARE INSERTED SUCCESSFULLY"<<endl;
    do{
        cout<<"********THIS PROGRAM IS GRAPH REPRESENTATION*****"<<endl;
        cout<<"1. INSERT NODE"<<endl;
        cout<<"2. INSERT EDGE"<<endl;
        cout<<"3. PRINT NODE-LIST"<<endl;
        cout<<"4. PRINT ADJACENCY-LIST"<<endl;
        cout<<"5. DELETE NODE"<<endl;
        cout<<"6. DELETE EDGE"<<endl;
        cout<<"7. CALCULATE DEGREE OF A NODE"<<endl;
        cout<<"8. CALCULATE INDEGREE OF A NODE"<<endl;
        cout<<"9. TOPOLOGICAL SORT"<<endl;
        cout<<"Enter your choice: ";
        cin>>choice;
        switch(choice){
            case 1:{
                cout<<"Enter the number of vertices: ";
                cin>>n;
```

```cpp
        while(n){
            cout<<"Enter vertex: ";
            cin>>c;
            insertNode(start,c);
            n--;
        }
        cout<<"ALL THE NODES ARE INSERTED SUCCESSFULLY\n"<<endl;
        break;
    }
    case 2:{
        cout<<"Enter the number of edges: ";
        cin>>edges;
        while(edges){
            char ch1,ch2;
            cout<<"Enter Vertices: "<<endl;
            cin>>ch1>>ch2;
            createEdge(start,ch1,ch2);
            edges--;
        }
        cout<<"ALL THE EDGES ARE INSERTED SUCCESSFULLY\n"<<endl;
        break;
    }
    case 3:{
        cout<<"Node list is: ";
        printNode(start);
        cout<<endl;
        break;
    }
    case 4:{
        cout<<"Enter the Element for Adjacencylist: ";
        cin>>c;
        cout<<"Adjacencylist is: "<<endl;
        printAdj(start,c);
        cout<<endl;
        break;
    }
    case 5:{
        cout<<"Enter the Node you want to delete: ";
        cin>>c;
        start=DeleteNode(start,c);
        cout<<"DELETE SUCCESSFULLY\n"<<endl;
        break;
    }
    case 6:{
        char ch1,ch2;
        cout<<"Enter the Edges you want to delete: ";
        cin>>ch1>>ch2;
        DeleteEdge(start,ch1,ch2);
```

```cpp
                    cout<<"DELETE SUCCESSFULLY\n"<<endl;
                    break;
                }
                case 7:{
                    // calculte degree
                    char c;
                    cout<<"Enter the vertix to find its Degree: ";
                    cin>>c;
                    Cal_degree(start,c);
                    cout<<endl;
                    break;
                }
                case 8:{
                    char c;
                    cout<<"Enter the vertix to find its InDegree: ";
                    cin>>c;
                    cout<<cal_indegree(start,c)<<endl<<endl;
                    break;
                }
                case 9:{
                    cout<<"AFTER SORTING: "<<endl;
                    topologicalSort(start);
                    break;
                }
                default:{
                    cout<<"Enter the valid choice!!!"<<endl;
                    break;
                }
            }
        cout<<"Do u wish to continue(y/n): ";
        cin>>ch;
        cout<<endl;
    }while(ch!='n');
    return 0;
}
// Adjacencylist is: a-> c,b,d,NULL
// b-> d,c,NULL
// c-> NULL
// d-> c,e,NULL
// e-> c,NULL
/*
            a----c
            |\  /|\
            | \/ | \
            | /\ |  e
            |/  \| /
            b----d/
*/
```

```
TERMINAL    PROBLEMS    OUTPUT    DEBUG CONSOLE

Enter the number of vertices: 5
Enter vertex: a
Enter vertex: b
Enter vertex: c
Enter vertex: d
Enter vertex: e
Enter the number of edges: 8
Enter Vertices:
a
b
Enter Vertices:
b
d
Enter Vertices:
d
c
Enter Vertices:
c
b
Enter Vertices:
a
c
Enter Vertices:
c
e
Enter Vertices:
e
d
Enter Vertices:
c
d
ALL THE EDGES ARE INSERTED SUCCESSFULLY
********THIS PROGRAM IS GRAPH REPRESENTATION*****
1.  INSERT NODE
2.  INSERT EDGE
3.  PRINT NODE-LIST
4.  PRINT ADJACENCY-LIST
```

```
c
d
ALL THE EDGES ARE INSERTED SUCCESSFULLY
********THIS PROGRAM IS GRAPH REPRESENTATION*****
1.  INSERT NODE
2.  INSERT EDGE
3.  PRINT NODE-LIST
4.  PRINT ADJACENCY-LIST
5.  DELETE NODE
6.  DELETE EDGE
7.  CALCULATE DEGREE OF A NODE
8.  CALCULATE INDEGREE OF A NODE
9.  TOPOLOGICAL SORT
Enter your choice: 7
Enter the vertix to find its Degree: a
Degree of a->  2

Do u wish to continue(y/n): y

********THIS PROGRAM IS GRAPH REPRESENTATION*****
1.  INSERT NODE
2.  INSERT EDGE
3.  PRINT NODE-LIST
4.  PRINT ADJACENCY-LIST
5.  DELETE NODE
6.  DELETE EDGE
7.  CALCULATE DEGREE OF A NODE
8.  CALCULATE INDEGREE OF A NODE
9.  TOPOLOGICAL SORT
Enter your choice: 7
Enter the vertix to find its Degree: c
Degree of c->  3

Do u wish to continue(y/n): n

PS C:\Users\anil kumar\Documents\anil\.vscode\DataSructure_in_nsut>
```