# Harsh_Kumar_MAN_106_Assignment_8

🌴 Order of the question : 2 1 3 4

- **Answer 2:**
  - Program :

```cpp
#include<iostream>
using namespace std;

class directed_graph{
private:
    int vertex;
    int edges;
    int** adl;

public:
    directed_graph(int);
    bool isEdge(int,int);
    void addEdge(int,int);
    int inDegree(int);
    int outDegree(int);
    void delEdge(int,int);
    int nEdges();
    void print();

};

directed_graph::directed_graph(int v){
    vertex = v;
    edges = 0;
    adl = new int* [vertex];
    for (int i=0; i<vertex; i++) adl[i] = new int[vertex];
    for (int i=0; i<vertex; i++){
        for (int j=0; j<vertex; j++) adl[i][j] = 0;
    }
}

void directed_graph::print(){
    cout << "\nAdjegency Matrix\n";
    for (int i=0; i<vertex; i++){
        for (int j=0; j<vertex; j++) cout << adl[i][j] << " ";
            cout << '\n';
    }
```

```
    cout << "Edges\n";
    for(int i=0; i<vertex; i++){
        for (int j=0; j< vertex; j++) if(adl[i][j]==1) cout << "(" << i+1 << " ," << j+1 << ")\n";
    }
    cout << "\n";
}

void directed_graph::addEdge(int a, int b){
    if (a-1 < vertex && b-1 < vertex) adl[a-1][b-1] = 1, edges++;
}

int directed_graph::inDegree(int node){
    int indeg=0;
    if (node > vertex) return -1;
    for(int i=0; i<vertex; i++) indeg+= adl[i][node-1];
    return indeg;
}

int directed_graph::outDegree(int node){
    int outdeg=0;
    if (node > vertex) return -1;
    for(int i=0; i<vertex; i++) outdeg+= adl[node-1][i];
    return outdeg;
}

bool directed_graph::isEdge(int a, int b){
    if (a < vertex && b < vertex && adl[a-1][b-1]==1) return true;
    else return false;
}

void directed_graph::delEdge(int a, int b){
    if (a < vertex && b < vertex) adl[a-1][b-1] = 0, edges--;
}
```

```
int directed_graph::nEdges(){
    return edges;
}

int main(){
    directed_graph D(4);

    D.addEdge(1,2);
    D.addEdge(2,3);
    D.addEdge(3,2);
    D.addEdge(4,1);
    D.addEdge(4,3);

    D.print();

    cout <<"inDegree of 1: " << D.inDegree(1) << " , outDegree of 1: " << D.outDegree(1) << endl;
    cout <<"inDegree of 2: " << D.inDegree(2) << " , outDegree of 2: " << D.outDegree(2) << endl;
    cout <<"inDegree of 3: " << D.inDegree(3) << " , outDegree of 3: " << D.outDegree(3) << endl;
    cout <<"inDegree of 4: " << D.inDegree(4) << " , outDegree of 4: " << D.outDegree(4) << endl;

    cout << "\n\nDeleting edge  (2,3)\n";
    D.delEdge(2,3);

    D.print();
}
```

- Output:

```
Adjegency Matrix
0 1 0 0
0 0 1 0
0 1 0 0
1 0 1 0
Edges
(1 ,2)
(2 ,3)
(3 ,2)
(4 ,1)
(4 ,3)

inDegree of 1: 1 , outDegree of 1: 1
inDegree of 2: 2 , outDegree of 2: 1
inDegree of 3: 2 , outDegree of 3: 1
inDegree of 4: 0 , outDegree of 4: 2


Deleting edge  (2,3)

Adjegency Matrix
0 1 0 0
0 0 0 0
0 1 0 0
1 0 1 0
Edges
(1 ,2)
(3 ,2)
(4 ,1)
(4 ,3)
```

- **Answer 1:**
  - Input :

```
D.addEdge(1,2)
D.addEdge(2,3)
D.addEdge(3,2)
D.addEdge(4,1)
D.addEdge(4,3)
```

  - Output:

```
Adjegency Matrix
0 1 0 0
0 0 1 0
0 1 0 0
1 0 1 0
Edges
(1 ,2)
(2 ,3)
(3 ,2)
(4 ,1)
(4 ,3)
```

- **Answer 3:**
  - Program :

```cpp
#include<iostream>
using namespace std;

class weighted_graph{
private:
    int vertex;
    int edges;
    int** adl;

public:
    weighted_graph(int);
    bool isEdge(int,int);
    void addEdge(int,int, int);
    int degree(int);
    void delEdge(int,int);
    int nEdges();
    void print();

};

weighted_graph::weighted_graph(int v){
    vertex = v;
    edges = 0;
    adl = new int* [vertex];
    for (int i=0; i<vertex; i++) adl[i] = new int[vertex];
    for (int i=0; i<vertex; i++){
        for (int j=0; j<vertex; j++) adl[i][j] = 0;
    }
}

void weighted_graph::print(){
    cout << "\nAdjegency Matrix\n";
    for (int i=0; i<vertex; i++){
        for (int j=0; j<vertex; j++) cout << adl[i][j] << " ";
            cout << '\n';
    }
```

```cpp
    cout << "Edges\n";
    for(int i=0; i<vertex; i++){
        for (int j=0; j< vertex; j++) if(adl[i][j]!=0) cout << "(" << i+1 << " ," << j+1 << " ," << adl[i][j] << ")
        cout << "\n";
    }
    cout << "\n";
}

void weighted_graph::addEdge(int a, int b, int w){
    if (a-1 < vertex && b-1 < vertex) adl[a-1][b-1] = adl[b-1][a-1] = w, edges++;
}

int weighted_graph::degree(int node){
    int indeg=0;
    if (node > vertex) return -1;
    for(int i=0; i<vertex; i++) if (adl[i][node-1] != 0) indeg++;
    return indeg;
}

bool weighted_graph::isEdge(int a, int b){
    if (a < vertex && b < vertex && adl[a-1][b-1]==1) return true;
    else return false;
}

void weighted_graph::delEdge(int a, int b){
    if (a < vertex && b < vertex) adl[a-1][b-1] = adl[b-1][a-1] = 0, edges--;
}

int weighted_graph::nEdges(){
    return edges;
}
```

```cpp
int main(){
    weighted_graph D(4);

    D.addEdge(1,2,6);
    D.addEdge(2,3,5);
    D.addEdge(4,1,2);
    D.addEdge(4,3,1);

    D.print();

    cout <<"Degree of 1: " << D.degree(1) << endl;
    cout <<"Degree of 2: " << D.degree(2) << endl;
    cout <<"Degree of 3: " << D.degree(3) << endl;
    cout <<"Degree of 4: " << D.degree(4) << endl;

    cout << "\n\nDeleting edge  (2,3)\n";
    D.delEdge(2,3);

    D.print();
}
```

- Output:

```
Adjegency Matrix
0 6 0 2
6 0 5 0
0 5 0 1
2 0 1 0
Edges
(1 ,2 ,6)(1 ,4 ,2)
(2 ,1 ,6)(2 ,3 ,5)
(3 ,2 ,5)(3 ,4 ,1)
(4 ,1 ,2)(4 ,3 ,1)

Degree of 1: 2
Degree of 2: 2
Degree of 3: 2
Degree of 4: 2


Deleting edge  (2,3)

Adjegency Matrix
0 6 0 2
6 0 0 0
0 0 0 1
2 0 1 0
Edges
(1 ,2 ,6)(1 ,4 ,2)
(2 ,1 ,6)
(3 ,4 ,1)
(4 ,1 ,2)(4 ,3 ,1)
```

- **Answer 4a:**
  - Program:

```cpp
#include<bits/stdc++.h>
using namespace std;

struct node{
    int vertex;
    node* next;
};


class directed_graph{
private:
    int vertex;
    int edges;
    node** adl;
public:
    directed_graph(int);
    bool isEdge(int , int);
    void addEdge(int,int);
    int inDegree(int);
    int outDegree(int);
    void delEdge(int,int);
    void print();
};

directed_graph::directed_graph(int v){
    vertex = v;
    edges = 0;
    adl = new node*[vertex];
    for(int i=0; i<vertex; i++) adl[i] = new node, adl[i]->vertex = i+1, adl[i]->next=NULL;
    cout << "Contructed\n";
}

```

```cpp
bool directed_graph::isEdge(int a, int b){
    auto temp = adl[a-1];
    for(; temp!=NULL; temp=temp->next) if (temp->vertex==b) return true;
    return false;
}

void directed_graph::addEdge(int a, int b){
    auto temp = adl[a-1];
    node* member;
    member = new node;
    member->vertex = b;
    member->next = NULL;

    while (temp->next!=NULL) temp=temp->next;
    temp->next = member;
}

void directed_graph::print(){
    for(int i=0; i<vertex; i++){
        auto temp = adl[i];
        temp = temp->next;
        while (temp!=NULL) {
            cout << "(" << i+1 << " ," << temp->vertex << ")";
            temp=temp->next;
        }
        cout << "\n";
    }
}
```

```cpp
int directed_graph::inDegree(int n){
    int indeg=0;
    for(int i=0; i<vertex; i++){
        if (i+1==n) continue;
        auto temp = adl[i];
        temp = temp->next;
        for (; temp!=NULL; temp=temp->next) if(temp->vertex==n) indeg++;
    }
    return indeg;
}

int directed_graph::outDegree(int n){
    int outdeg = 0;
    auto temp = adl[n-1];
    temp = temp->next;
    for (; temp!=NULL; temp=temp->next) outdeg++;
    return outdeg;
}

void directed_graph::delEdge(int a, int b){
    auto temp = adl[a-1];
    for (; temp->next!=NULL; temp=temp->next) {
        if(temp->next->vertex==b){
            cout << "Edge found\n";
            auto curr = temp->next;
            temp->next = curr->next;
            delete curr;
            cout << "Edge deleted\n";
            return ;
        }
    }
    cout << "Edge not found\n";
    return ;
}
```

```cpp
int main(){

    directed_graph D(4);

    D.addEdge(1,2);
    D.addEdge(1,3);
    D.addEdge(1,4);
    D.addEdge(2,3);
    D.addEdge(3,1);
    D.addEdge(4,1);

    D.print();

    cout << "\n" << (D.isEdge(3,2) ? "YES\n" : "NO\n");

    cout <<"inDegree of 1: " << D.inDegree(1) << " , outDegree of 1: " << D.outDegree(1) << end
    cout <<"inDegree of 2: " << D.inDegree(2) << " , outDegree of 2: " << D.outDegree(2) << end
    cout <<"inDegree of 3: " << D.inDegree(3) << " , outDegree of 3: " << D.outDegree(3) << end
    cout <<"inDegree of 4: " << D.inDegree(4) << " , outDegree of 4: " << D.outDegree(4) << end

    cout << "\n\nDeleting edge (3,1)\n";
    D.delEdge(3,1);

    D.print();

    return 0;
}
```

- Output:

```
Contructed
(1 ,2)(1 ,3)(1 ,4)
(2 ,3)
(3 ,1)
(4 ,1)

NO
inDegree of 1: 2 , outDegree of 1: 3
inDegree of 2: 1 , outDegree of 2: 1
inDegree of 3: 2 , outDegree of 3: 1
inDegree of 4: 1 , outDegree of 4: 1


Deleting edge (3,1)
Edge found
Edge deleted
(1 ,2)(1 ,3)(1 ,4)
(2 ,3)

(4 ,1)
```

- **Answer 4b:**
  - Program :

```cpp
#include<bits/stdc++.h>
using namespace std;

struct node{
    int vertex;
    node* next;
};


class undirected_graph{
private:
    int vertex;
    int edges;
    node** adl;
public:
    undirected_graph(int);
    bool isEdge(int , int);
    void addEdge(int,int);
    int degree(int);
    void delEdge(int,int);
    void print();
};

undirected_graph::undirected_graph(int v){
    vertex = v;
    edges = 0;
    adl = new node*[vertex];
    for(int i=0; i<vertex; i++) adl[i] = new node, adl[i]->vertex = i+1, adl[i]->next=NULL;
    cout << "Contructed\n";
}

bool undirected_graph::isEdge(int a, int b){
    auto temp = adl[a-1];
    for(; temp!=NULL; temp=temp->next) if (temp->vertex==b) return true;
    return false;
}
```

```cpp
void undirected_graph::addEdge(int a, int b){
    auto temp = adl[a-1];
    node* member_1;
    member_1 = new node;
    member_1->vertex = b;
    member_1->next = NULL;

    while (temp->next!=NULL) temp=temp->next;
    temp->next = member_1;

    temp = adl[b-1];
    node* member_2;
    member_2 = new node;
    member_2->vertex = a;
    member_2->next = NULL;

    while (temp->next!=NULL) temp=temp->next;
    temp->next = member_2;
}

void undirected_graph::print(){
    for(int i=0; i<vertex; i++){
        auto temp = adl[i];
        temp = temp->next;
        while (temp!=NULL) {
            cout << "(" << i+1 << " ," << temp->vertex << ")";
            temp=temp->next;
        }
        cout << "\n";
    }
}
```

```cpp
int undirected_graph::degree(int n){
    int deg = 0;
    auto temp = adl[n-1];
    temp = temp->next;
    for (; temp!=NULL; temp=temp->next) deg++;
    return deg;
}

void undirected_graph::delEdge(int a, int b){
    auto temp = adl[a-1];
    for (; temp->next!=NULL; temp=temp->next) {
        if(temp->next->vertex==b){
            auto curr = temp->next;
            temp->next = curr->next;
            delete curr;
        }
    }
    temp = adl[b-1];
    for (; temp->next!=NULL; temp=temp->next) {
        if(temp->next->vertex==a){
            auto curr = temp->next;
            temp->next = curr->next;
            delete curr;
        }
    }
}
```

```cpp
int main(){

    undirected_graph D(4);

    D.addEdge(1,2);
    D.addEdge(1,3);
    D.addEdge(4,1);
    D.addEdge(2,3);


    D.print();

    cout << "\n" << (D.isEdge(3,2) ? "YES\n" : "NO\n");

    cout <<"Degree of 1: " << D.degree(1) << endl;
    cout <<"Degree of 2: " << D.degree(2) << endl;
    cout <<"Degree of 3: " << D.degree(3) << endl;
    cout <<"Degree of 4: " << D.degree(4) << endl;

    cout << "\n\nDeleting edge (3,1)\n";
    D.delEdge(3,1);

    D.print();

    return 0;
}
```

- Output:

```
Contructed
(1 ,2)(1 ,3)(1 ,4)
(2 ,1)(2 ,3)
(3 ,1)(3 ,2)
(4 ,1)

YES
Degree of 1: 3
Degree of 2: 2
Degree of 3: 2
Degree of 4: 1


Deleting edge (3,1)
(1 ,2)(1 ,4)
(2 ,1)(2 ,3)
(3 ,2)
(4 ,1)
```

- **Answer 4c:**
  - Program:

```cpp
#include<bits/stdc++.h>
using namespace std;

struct node{
    int vertex;
    int weight;
    node* next;
};


class weighted_graph{
private:
    int vertex;
    int edges;
    node** adl;
public:
    weighted_graph(int);
    bool isEdge(int , int);
    void addEdge(int,int,int);
    int degree(int);
    void delEdge(int,int);
    void print();
};

weighted_graph::weighted_graph(int v){
    vertex = v;
    edges = 0;
    adl = new node*[vertex];
    for(int i=0; i<vertex; i++) adl[i] = new node, adl[i]->vertex = i+1, adl[i]->next=NULL;
    cout << "Contructed\n";
}
```

```cpp
bool weighted_graph::isEdge(int a, int b){
    auto temp = adl[a-1];
    for(; temp!=NULL; temp=temp->next) if (temp->vertex==b) return true;
    return false;
}

void weighted_graph::addEdge(int a, int b, int w){
    auto temp = adl[a-1];
    node* member_1;
    member_1 = new node;
    member_1->vertex = b;
    member_1->weight = w;
    member_1->next = NULL;

    while (temp->next!=NULL) temp=temp->next;
    temp->next = member_1;

    temp = adl[b-1];
    node* member_2;
    member_2 = new node;
    member_2->vertex = a;
    member_2->weight = w;
    member_2->next = NULL;

    while (temp->next!=NULL) temp=temp->next;
    temp->next = member_2;
}
```

```cpp
void weighted_graph::print(){
    for(int i=0; i<vertex; i++){
        auto temp = adl[i];
        temp = temp->next;
        while (temp!=NULL) {
            cout << "(" << i+1 << " ," << temp->vertex << " ," << temp->weight << ")";
            temp=temp->next;
        }
        cout << "\n";
    }
}


int weighted_graph::degree(int n){
    int deg = 0;
    auto temp = adl[n-1];
    temp = temp->next;
    for (; temp!=NULL; temp=temp->next) deg++;
    return deg;
}

void weighted_graph::delEdge(int a, int b){
    auto temp = adl[a-1];
    for (; temp->next!=NULL; temp=temp->next) {
        if(temp->next->vertex==b){
            auto curr = temp->next;
            temp->next = curr->next;
            delete curr;
        }
    }
    temp = adl[b-1];
    for (; temp->next!=NULL; temp=temp->next) {
        if(temp->next->vertex==a){
            auto curr = temp->next;
            temp->next = curr->next;
            delete curr;
        }
```

```cpp
int main(){

    weighted_graph D(4);

    D.addEdge(1,2,5);
    D.addEdge(1,3,6);
    D.addEdge(4,1,2);
    D.addEdge(2,3,8);


    D.print();

    cout << "\n" << (D.isEdge(3,2) ? "YES\n" : "NO\n");

    cout <<"Degree of 1: " << D.degree(1) << endl;
    cout <<"Degree of 2: " << D.degree(2) << endl;
    cout <<"Degree of 3: " << D.degree(3) << endl;
    cout <<"Degree of 4: " << D.degree(4) << endl;

    cout << "\n\nDeleting edge (3,1)\n";
    D.delEdge(3,1);

    D.print();

    return 0;
}
```

- Output:

```
Contructed
(1 ,2 ,5)(1 ,3 ,6)(1 ,4 ,2)
(2 ,1 ,5)(2 ,3 ,8)
(3 ,1 ,6)(3 ,2 ,8)
(4 ,1 ,2)

YES
Degree of 1: 3
Degree of 2: 2
Degree of 3: 2
Degree of 4: 1


Deleting edge (3,1)
(1 ,2 ,5)(1 ,4 ,2)
(2 ,1 ,5)(2 ,3 ,8)
(3 ,2 ,8)
(4 ,1 ,2)
```