

# Harsh\_Kumar\_MAN\_106\_Assignment\_8



Order of the question : 2 1 3 4

- **Answer 2:**

- Program :

```
1  #include<iostream>
2  using namespace std;
3
4  class directed_graph{
5  private:
6      int vertex;
7      int edges;
8      int** adl;
9
10 public:
11     directed_graph(int);
12     bool isEdge(int,int);
13     void addEdge(int,int);
14     int inDegree(int);
15     int outDegree(int);
16     void delEdge(int,int);
17     int nEdges();
18     void print();
19
20 };
21
22 directed_graph::directed_graph(int v){
23     vertex = v;
24     edges = 0;
25     adl = new int* [vertex];
26     for (int i=0; i<vertex; i++) adl[i] = new int[vertex];
27     for (int i=0; i<vertex; i++){
28         for (int j=0; j<vertex; j++) adl[i][j] = 0;
29     }
30 }
31
32 void directed_graph::print(){
33     cout << "\nAdjacency Matrix\n";
34     for (int i=0; i<vertex; i++){
35         for (int j=0; j<vertex; j++) cout << adl[i][j] << " ";
36         cout << '\n';
37     }
```

```

38
39     cout << "Edges\n";
40     for(int i=0; i<vertex; i++){
41         for (int j=0; j< vertex; j++) if(adl[i][j]==1) cout << "(" << i+1 << " , " << j+1 << ")\n";
42     }
43     cout << "\n";
44 }
45
46 void directed_graph::addEdge(int a, int b){
47     if (a-1 < vertex && b-1 < vertex) adl[a-1][b-1] = 1, edges++;
48 }
49
50 int directed_graph::inDegree(int node){
51     int indeg=0;
52     if (node > vertex) return -1;
53     for(int i=0; i<vertex; i++) indeg+= adl[i][node-1];
54     return indeg;
55 }
56
57 int directed_graph::outDegree(int node){
58     int outdeg=0;
59     if (node > vertex) return -1;
60     for(int i=0; i<vertex; i++) outdeg+= adl[node-1][i];
61     return outdeg;
62 }
63
64 bool directed_graph::isEdge(int a, int b){
65     if (a < vertex && b < vertex && adl[a-1][b-1]==1) return true;
66     else return false;
67 }
68
69 void directed_graph::delEdge(int a, int b){
70     if (a < vertex && b < vertex) adl[a-1][b-1] = 0, edges--;
71 }
72

```

```

73 int directed_graph::nEdges(){
74     return edges;
75 }
76
77 int main(){
78     directed_graph D(4);
79
80     D.addEdge(1,2);
81     D.addEdge(2,3);
82     D.addEdge(3,2);
83     D.addEdge(4,1);
84     D.addEdge(4,3);
85
86     D.print();
87
88     cout <<"inDegree of 1: " << D.inDegree(1) << " , outDegree of 1: " << D.outDegree(1) << endl;
89     cout <<"inDegree of 2: " << D.inDegree(2) << " , outDegree of 2: " << D.outDegree(2) << endl;
90     cout <<"inDegree of 3: " << D.inDegree(3) << " , outDegree of 3: " << D.outDegree(3) << endl;
91     cout <<"inDegree of 4: " << D.inDegree(4) << " , outDegree of 4: " << D.outDegree(4) << endl;
92
93     cout << "\n\nDeleting edge (2,3)\n";
94     D.delEdge(2,3);
95
96     D.print();
97 }

```

- Output:

```

Adjacency 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)

```

Adjacency 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:

```

Adjacency 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 :

```

1  #include<iostream>
2  using namespace std;
3
4  class weighted_graph{
5  private:
6      int vertex;
7      int edges;
8      int** adl;
9
10 public:
11     weighted_graph(int);
12     bool isEdge(int,int);
13     void addEdge(int,int, int);
14     int degree(int);
15     void delEdge(int,int);
16     int nEdges();
17     void print();
18
19 };
20
21 weighted_graph::weighted_graph(int v){
22     vertex = v;
23     edges = 0;
24     adl = new int* [vertex];
25     for (int i=0; i<vertex; i++) adl[i] = new int[vertex];
26     for (int i=0; i<vertex; i++){
27         for (int j=0; j<vertex; j++) adl[i][j] = 0;
28     }
29 }
30
31 void weighted_graph::print(){
32     cout << "\nAdjacency Matrix\n";
33     for (int i=0; i<vertex; i++){
34         for (int j=0; j<vertex; j++) cout << adl[i][j] << " ";
35         cout << '\n';
36     }
37
38     cout << "Edges\n";
39     for(int i=0; i<vertex; i++){
40         for (int j=0; j< vertex; j++) if(adl[i][j]!=0) cout << "(" << i+1 << " , " << j+1 << " , " << adl[i][j] << " )";
41         cout << "\n";
42     }
43     cout << "\n";
44 }
45
46 void weighted_graph::addEdge(int a, int b, int w){
47     if (a-1 < vertex && b-1 < vertex) adl[a-1][b-1] = adl[b-1][a-1] = w, edges++;
48 }
49
50 int weighted_graph::degree(int node){
51     int indeg=0;
52     if (node > vertex) return -1;
53     for(int i=0; i<vertex; i++) if (adl[i][node-1] != 0) indeg++;
54     return indeg;
55 }
56
57 bool weighted_graph::isEdge(int a, int b){
58     if (a < vertex && b < vertex && adl[a-1][b-1]==1) return true;
59     else return false;
60 }
61
62 void weighted_graph::delEdge(int a, int b){
63     if (a < vertex && b < vertex) adl[a-1][b-1] = adl[b-1][a-1] = 0, edges--;
64 }
65
66 int weighted_graph::nEdges(){
67     return edges;
68 }
69

```

```

70  int main(){
71      weighted_graph D(4);
72
73      D.addEdge(1,2,6);
74      D.addEdge(2,3,5);
75      D.addEdge(4,1,2);
76      D.addEdge(4,3,1);
77
78      D.print();
79
80      cout << "Degree of 1: " << D.degree(1) << endl;
81      cout << "Degree of 2: " << D.degree(2) << endl;
82      cout << "Degree of 3: " << D.degree(3) << endl;
83      cout << "Degree of 4: " << D.degree(4) << endl;
84
85      cout << "\n\nDeleting edge (2,3)\n";
86      D.delEdge(2,3);
87
88      D.print();
89  }

```

- 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:

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  struct node{
5      int vertex;
6      node* next;
7  };
8
9
10 class directed_graph{
11 private:
12     int vertex;
13     int edges;
14     node** adl;
15 public:
16     directed_graph(int);
17     bool isEdge(int , int);
18     void addEdge(int,int);
19     int inDegree(int);
20     int outDegree(int);
21     void delEdge(int,int);
22     void print();
23 };
24
25 directed_graph::directed_graph(int v){
26     vertex = v;
27     edges = 0;
28     adl = new node*[vertex];
29     for(int i=0; i<vertex; i++) adl[i] = new node, adl[i]→vertex = i+1, adl[i]→next=NULL;
30     cout << "Constructed\n";
31 }
32

```

```

33 bool directed_graph::isEdge(int a, int b){
34     auto temp = adl[a-1];
35     for(; temp≠NULL; temp=temp→next) if (temp→vertex==b) return true;
36     return false;
37 }
38
39 void directed_graph::addEdge(int a, int b){
40     auto temp = adl[a-1];
41     node* member;
42     member = new node;
43     member→vertex = b;
44     member→next = NULL;
45
46     while (temp→next≠NULL) temp=temp→next;
47     temp→next = member;
48 }
49
50 void directed_graph::print(){
51     for(int i=0; i<vertex; i++){
52         auto temp = adl[i];
53         temp = temp→next;
54         while (temp≠NULL) {
55             cout << "(" << i+1 << " , " << temp→vertex << ")";
56             temp=temp→next;
57         }
58         cout << "\n";
59     }
60 }

```

```

62  int directed_graph::inDegree(int n){
63      int indeg=0;
64      for(int i=0; i<vertex; i++){
65          if (i+1==n) continue;
66          auto temp = adl[i];
67          temp = temp->next;
68          for (; temp!=NULL; temp=temp->next) if(temp->vertex==n) indeg++;
69      }
70      return indeg;
71  }
72
73  int directed_graph::outDegree(int n){
74      int outdeg = 0;
75      auto temp = adl[n-1];
76      temp = temp->next;
77      for (; temp!=NULL; temp=temp->next) outdeg++;
78      return outdeg;
79  }
80
81  void directed_graph::delEdge(int a, int b){
82      auto temp = adl[a-1];
83      for (; temp->next!=NULL; temp=temp->next) {
84          if(temp->next->vertex==b){
85              cout << "Edge found\n";
86              auto curr = temp->next;
87              temp->next = curr->next;
88              delete curr;
89              cout << "Edge deleted\n";
90              return ;
91          }
92      }
93      cout << "Edge not found\n";
94      return ;
95  }

```

```

97  int main(){
98
99      directed_graph D(4);
100
101      D.addEdge(1,2);
102      D.addEdge(1,3);
103      D.addEdge(1,4);
104      D.addEdge(2,3);
105      D.addEdge(3,1);
106      D.addEdge(4,1);
107
108      D.print();
109
110      cout << "\n" << (D.isEdge(3,2) ? "YES\n" : "NO\n");
111
112      cout << "inDegree of 1: " << D.inDegree(1) << " , outDegree of 1: " << D.outDegree(1) << endl;
113      cout << "inDegree of 2: " << D.inDegree(2) << " , outDegree of 2: " << D.outDegree(2) << endl;
114      cout << "inDegree of 3: " << D.inDegree(3) << " , outDegree of 3: " << D.outDegree(3) << endl;
115      cout << "inDegree of 4: " << D.inDegree(4) << " , outDegree of 4: " << D.outDegree(4) << endl;
116
117      cout << "\n\nDeleting edge (3,1)\n";
118      D.delEdge(3,1);
119
120      D.print();
121
122      return 0;
123  }

```

- Output:

```
Constructed
(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 :



```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  struct node{
5      int vertex;
6      node* next;
7  };
8
9
10 class undirected_graph{
11 private:
12     int vertex;
13     int edges;
14     node** adl;
15 public:
16     undirected_graph(int);
17     bool isEdge(int , int);
18     void addEdge(int,int);
19     int degree(int);
20     void delEdge(int,int);
21     void print();
22 };
23
24 undirected_graph::undirected_graph(int v){
25     vertex = v;
26     edges = 0;
27     adl = new node*[vertex];
28     for(int i=0; i<vertex; i++) adl[i] = new node, adl[i]→vertex = i+1, adl[i]→next=NULL;
29     cout << "Constructed\n";
30 }
31
32 bool undirected_graph::isEdge(int a, int b){
33     auto temp = adl[a-1];
34     for(; temp≠NULL; temp=temp→next) if (temp→vertex==b) return true;
35     return false;
36 }

```

```

38 void undirected_graph::addEdge(int a, int b){
39     auto temp = adl[a-1];
40     node* member_1;
41     member_1 = new node;
42     member_1->vertex = b;
43     member_1->next = NULL;
44
45     while (temp->next!=NULL) temp=temp->next;
46     temp->next = member_1;
47
48     temp = adl[b-1];
49     node* member_2;
50     member_2 = new node;
51     member_2->vertex = a;
52     member_2->next = NULL;
53
54     while (temp->next!=NULL) temp=temp->next;
55     temp->next = member_2;
56 }
57
58 void undirected_graph::print(){
59     for(int i=0; i<vertex; i++){
60         auto temp = adl[i];
61         temp = temp->next;
62         while (temp!=NULL) {
63             cout << "(" << i+1 << " , " << temp->vertex << ")";
64             temp=temp->next;
65         }
66         cout << "\n";
67     }
68 }

```

```

71 int undirected_graph::degree(int n){
72     int deg = 0;
73     auto temp = adl[n-1];
74     temp = temp->next;
75     for (; temp!=NULL; temp=temp->next) deg++;
76     return deg;
77 }
78
79 void undirected_graph::delEdge(int a, int b){
80     auto temp = adl[a-1];
81     for (; temp->next!=NULL; temp=temp->next) {
82         if(temp->next->vertex==b){
83             auto curr = temp->next;
84             temp->next = curr->next;
85             delete curr;
86         }
87     }
88     temp = adl[b-1];
89     for (; temp->next!=NULL; temp=temp->next) {
90         if(temp->next->vertex==a){
91             auto curr = temp->next;
92             temp->next = curr->next;
93             delete curr;
94         }
95     }
96 }
97

```

```

98  int main(){
99
100     undirected_graph D(4);
101
102     D.addEdge(1,2);
103     D.addEdge(1,3);
104     D.addEdge(4,1);
105     D.addEdge(2,3);
106
107
108     D.print();
109
110     cout << "\n" << (D.isEdge(3,2) ? "YES\n" : "NO\n");
111
112     cout << "Degree of 1: " << D.degree(1) << endl;
113     cout << "Degree of 2: " << D.degree(2) << endl;
114     cout << "Degree of 3: " << D.degree(3) << endl;
115     cout << "Degree of 4: " << D.degree(4) << endl;
116
117     cout << "\n\nDeleting edge (3,1)\n";
118     D.delEdge(3,1);
119
120     D.print();
121
122     return 0;
123 }
124

```

- Output:

```

Constructed
(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:

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  struct node{
5      int vertex;
6      int weight;
7      node* next;
8  };
9
10
11 class weighted_graph{
12 private:
13     int vertex;
14     int edges;
15     node** adl;
16 public:
17     weighted_graph(int);
18     bool isEdge(int , int);
19     void addEdge(int,int,int);
20     int degree(int);
21     void delEdge(int,int);
22     void print();
23 };
24
25 weighted_graph::weighted_graph(int v){
26     vertex = v;
27     edges = 0;
28     adl = new node*[vertex];
29     for(int i=0; i<vertex; i++) adl[i] = new node, adl[i]→vertex = i+1, adl[i]→next=NULL;
30     cout << "Constructed\n";
31 }

```

```

33 bool weighted_graph::isEdge(int a, int b){
34     auto temp = adl[a-1];
35     for(; temp≠NULL; temp=temp→next) if (temp→vertex==b) return true;
36     return false;
37 }
38
39 void weighted_graph::addEdge(int a, int b, int w){
40     auto temp = adl[a-1];
41     node* member_1;
42     member_1 = new node;
43     member_1→vertex = b;
44     member_1→weight = w;
45     member_1→next = NULL;
46
47     while (temp→next≠NULL) temp=temp→next;
48     temp→next = member_1;
49
50     temp = adl[b-1];
51     node* member_2;
52     member_2 = new node;
53     member_2→vertex = a;
54     member_2→weight = w;
55     member_2→next = NULL;
56
57     while (temp→next≠NULL) temp=temp→next;
58     temp→next = member_2;
59 }

```

```

61 void weighted_graph::print(){
62     for(int i=0; i<vertex; i++){
63         auto temp = adl[i];
64         temp = temp->next;
65         while (temp!=NULL) {
66             cout << "(" << i+1 << " , " << temp->vertex << " , " << temp->weight << ")";
67             temp=temp->next;
68         }
69         cout << "\n";
70     }
71 }
72
73
74 int weighted_graph::degree(int n){
75     int deg = 0;
76     auto temp = adl[n-1];
77     temp = temp->next;
78     for (; temp!=NULL; temp=temp->next) deg++;
79     return deg;
80 }
81
82 void weighted_graph::delEdge(int a, int b){
83     auto temp = adl[a-1];
84     for (; temp->next!=NULL; temp=temp->next) {
85         if(temp->next->vertex==b){
86             auto curr = temp->next;
87             temp->next = curr->next;
88             delete curr;
89         }
90     }
91     temp = adl[b-1];
92     for (; temp->next!=NULL; temp=temp->next) {
93         if(temp->next->vertex==a){
94             auto curr = temp->next;
95             temp->next = curr->next;
96             delete curr;
97         }
98     }

```

```

101 int main(){
102
103     weighted_graph D(4);
104
105     D.addEdge(1,2,5);
106     D.addEdge(1,3,6);
107     D.addEdge(4,1,2);
108     D.addEdge(2,3,8);
109
110
111     D.print();
112
113     cout << "\n" << (D.isEdge(3,2) ? "YES\n" : "NO\n");
114
115     cout << "Degree of 1: " << D.degree(1) << endl;
116     cout << "Degree of 2: " << D.degree(2) << endl;
117     cout << "Degree of 3: " << D.degree(3) << endl;
118     cout << "Degree of 4: " << D.degree(4) << endl;
119
120     cout << "\n\nDeleting edge (3,1)\n";
121     D.delEdge(3,1);
122
123     D.print();
124
125     return 0;
126 }

```

- Output:

```
..
Constructed
(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)