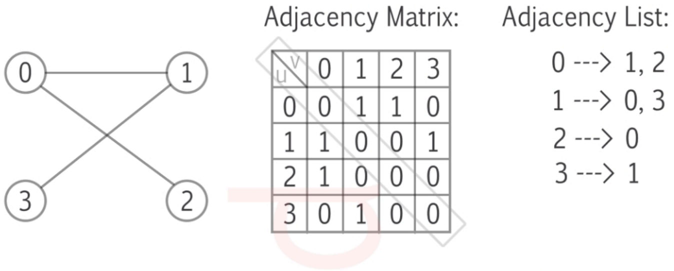
Graph Representation (Adjacency Matrix)

Undirected Unweighted graph



Adjacency Matrix:

u \ v	0	1	2	3
0	0	1	1	0
1	1	0	0	1
2	1	0	0	0
3	0	1	0	0

Adjacency List:

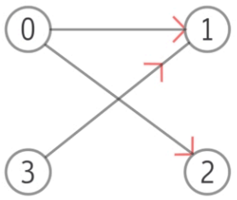
0 --->	1, 2
1 --->	0, 3
2 --->	0
3 --->	1

u is the current vertex, v is the verticies connected (have an edge) to u

```
// C++ program to demonstrate Adjacency Matrix
// representation of undirected and unweighted graph
#include <iostream>
#include <vector>
using namespace std;

void addEdge(vector<vector<int>> &mat, int i, int j)
{
   mat[i][j] = 1;
   mat[j][i] = 1; // Since the graph is undirected
}

void displayMatrix(vector<vector<int>> &mat)
{
   int V = mat.size();
   for (int i = 0; i < V; i++)
   {
      for (int j = 0; j < V; j++)
         cout << mat[i][j] << " ";
      cout << endl;
   }
}

int main()
{

   // Create a graph with 4 vertices and no edges
   // Note that all values are initialized as 0
   int V = 4;
   vector<vector<int>> mat(V, vector<int>(V, 0));
```

```
    // Now add edges one by one
    addEdge(mat, 0, 1);
    addEdge(mat, 0, 2);
    addEdge(mat, 1, 2);
    addEdge(mat, 2, 3);

    /* Alternatively we can also create using below
       code if we know all edges in advance

     vector<vector<int>> mat = {{ 0, 1, 0, 0 },
                                { 1, 0, 1, 0 },
                                { 0, 1, 0, 1 },
                                { 0, 0, 1, 0 } }; */

    cout << "Adjacency Matrix Representation" << endl;
    displayMatrix(mat);

    return 0;
}
```

Directed unweighted graph



Adjacency Matrix:

u\v	0	1	2	3
0	0	1	1	0
1	0	0	0	0
2	0	0	0	0
3	0	1	0	0

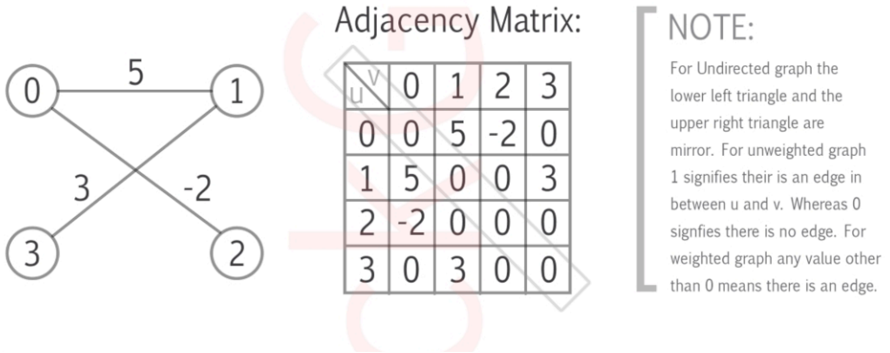
Adjacency List:

0 ---> 1, 2
1 --->
2 --->
3 ---> 1

```
// C++ program to demonstrate Adjacency Matrix
// representation of directed and unweighted graph
#include <iostream>
#include <vector>
using namespace std;

void addEdge(vector<vector<int>> &mat, int i, int j)
{
    mat[i][j] = 1; // Only add edge from i to j since the graph is directed
}

void displayMatrix(vector<vector<int>> &mat)
{
    int V = mat.size();
    for (int i = 0; i < V; i++)
    {
```

```
    for (int j = 0; j < V; j++)
      cout << mat[i][j] << " ";
    cout << endl;
  }
}

int main()
{
  // Create a graph with 4 vertices and no edges
  // Note that all values are initialized as 0
  int V = 4;
  vector<vector<int>> mat(V, vector<int>(V, 0));

  // Now add edges one by one
  addEdge(mat, 0, 1);
  addEdge(mat, 0, 2);
  addEdge(mat, 1, 2);
  addEdge(mat, 2, 3);

  /* Alternatively we can also create using below
    code if we know all edges in advance

   vector<vector<int>> mat = {{ 0, 1, 1, 0 },
                { 0, 0, 1, 0 },
                { 0, 0, 0, 1 },
                { 0, 0, 0, 0 } }; */

  cout << "Adjacency Matrix Representation" << endl;
  displayMatrix(mat);

  return 0;
}
```

Undirected weighted graph



Adjacency Matrix:

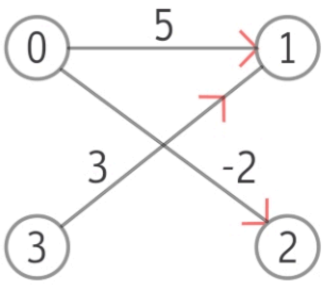
u\v	0	1	2	3
0	0	5	-2	0
1	5	0	0	3
2	-2	0	0	0
3	0	3	0	0

NOTE:

For Undirected graph the lower left triangle and the upper right triangle are mirror. For unweighted graph 1 signifies their is an edge in between u and v. Whereas 0 signfies there is no edge. For weighted graph any value other than 0 means there is an edge.

```
// C++ program to demonstrate Adjacency Matrix
// representation of undirected and weighted graph
#include <iostream>
#include <vector>
using namespace std;

void addEdge(vector<vector<int>> &mat, int i, int j, int weight)
{
  mat[i][j] = weight; // Add edge from i to j with the given weight
  mat[j][i] = weight; // Since the graph is undirected, add edge from j to i with the same weight
}

void displayMatrix(vector<vector<int>> &mat)
{
  int V = mat.size();
  for (int i = 0; i < V; i++)
  {
    for (int j = 0; j < V; j++)
      cout << mat[i][j] << " ";
    cout << endl;
  }
}

int main()
{
  // Create a graph with 4 vertices and no edges
  // Note that all values are initialized as 0
  int V = 4;
  vector<vector<int>> mat(V, vector<int>(V, 0));

  // Now add edges one by one with weights
  addEdge(mat, 0, 1, 2);
  addEdge(mat, 0, 2, 4);
  addEdge(mat, 1, 2, 1);
  addEdge(mat, 2, 3, 3);

  /* Alternatively we can also create using below
     code if we know all edges and weights in advance

   vector<vector<int>> mat = {{ 0, 2, 4, 0 },
                              { 2, 0, 1, 0 },
                              { 4, 1, 0, 3 },
                              { 0, 0, 3, 0 } }; */

  cout << "Adjacency Matrix Representation" << endl;
  displayMatrix(mat);

  return 0;
}
```

Directed and weighted graph



Adjacency Matrix:

u\v	0	1	2	3
0	0	5	-2	0
1	0	0	0	0
2	0	0	0	0
3	0	3	0	0

NOTE:

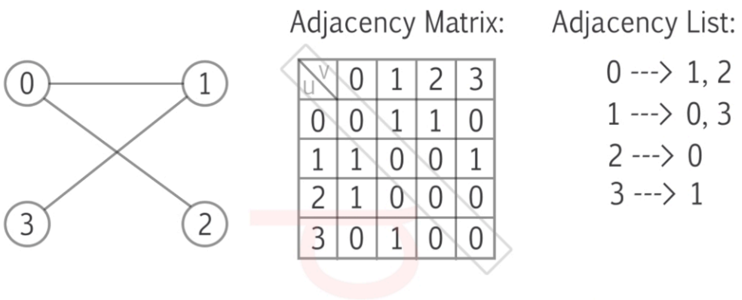
There are no outgoing edge from 1 and 2, only incoming edges. So their outdegree is zero. It is possible to go from 0 to 1 but it is not possible to go from 1 to 0.

```
// C++ program to demonstrate Adjacency Matrix
// representation of directed and weighted graph
#include <iostream>
#include <vector>
using namespace std;

void addEdge(vector<vector<int>> &mat, int i, int j, int weight)
{
   mat[i][j] = weight; // Add edge from i to j with the given weight since the graph is directed
}

void displayMatrix(vector<vector<int>> &mat)
{
   int V = mat.size();
   for (int i = 0; i < V; i++)
   {
      for (int j = 0; j < V; j++)
         cout << mat[i][j] << " ";
      cout << endl;
   }
}

int main()
{
   // Create a graph with 4 vertices and no edges
   // Note that all values are initialized as 0
   int V = 4;
   vector<vector<int>> mat(V, vector<int>(V, 0));

   // Now add edges one by one with weights
   addEdge(mat, 0, 1, 2);
   addEdge(mat, 0, 2, 4);
```

```
    addEdge(mat, 1, 2, 1);
    addEdge(mat, 2, 3, 3);

    /* Alternatively we can also create using below
       code if we know all edges and weights in advance

     vector<vector<int>> mat = {{ 0, 2, 4, 0 },
                                 { 0, 0, 1, 0 },
                                 { 0, 0, 0, 3 },
                                 { 0, 0, 0, 0 } }; */

    cout << "Adjacency Matrix Representation" << endl;
    displayMatrix(mat);

    return 0;
}
```

Graph Representation (Adjacency List)

Undirected Unweighted graph



Adjacency Matrix:

u\v	0	1	2	3
0	0	1	1	0
1	1	0	0	1
2	1	0	0	0
3	0	1	0	0

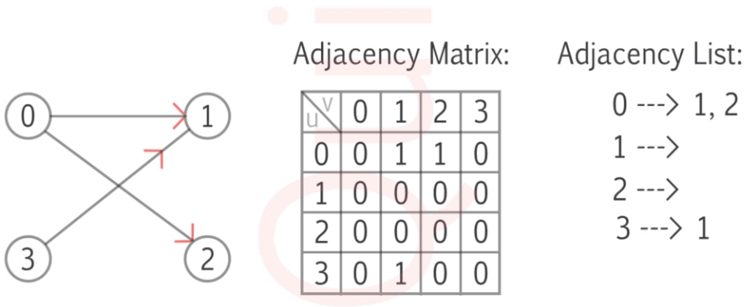
Adjacency List:

0 ---> 1, 2
1 ---> 0, 3
2 ---> 0
3 ---> 1

u is the current vertex, v is the verticies connected (have an edge) to u

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

// Function to add an edge between two vertices
void addEdge(vector<vector<int>>& adj, int i, int j) {
    adj[i].push_back(j);
    adj[j].push_back(i); // Undirected
}

// Function to display the adjacency list
void displayAdjList(const vector<vector<int>>& adj) {
    for (int i = 0; i < adj.size(); i++) {
       cout << i << ": "; // Print the vertex
       for (int j : adj[i]) {
          cout << j << " "; // Print its adjacent
       }
       cout << endl;
    }
}

// Main function
int main() {
    // Create a graph with 4 vertices and no edges
    int V = 4;
    vector<vector<int>> adj(V);

    // Now add edges one by one
    addEdge(adj, 0, 1);
    addEdge(adj, 0, 2);
    addEdge(adj, 1, 2);
    addEdge(adj, 2, 3);

    cout << "Adjacency List Representation:" << endl;
    displayAdjList(adj);

    return 0;
}
```

Directed unweighted graph



Adjacency Matrix:

u\v	0	1	2	3
0	0	1	1	0
1	0	0	0	0
2	0	0	0	0
3	0	1	0	0

Adjacency List:

0 ---> 1, 2
1 --->
2 --->
3 ---> 1

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

// Function to add a directed edge from vertex i to vertex j
void addEdge(vector<vector<int>>& adj, int i, int j) {
   adj[i].push_back(j); // Only add the edge from i to j (directed)
}
```

```
// Function to display the adjacency list
void displayAdjList(const vector<vector<int>>& adj) {
    for (int i = 0; i < adj.size(); i++) {
        cout << i << ": "; // Print the vertex
        for (int j : adj[i]) {
            cout << j << " "; // Print its adjacent
        }
        cout << endl;
    }
}

// Main function
int main() {
    // Create a graph with 4 vertices and no edges
    int V = 4;
    vector<vector<int>> adj(V);

    // Now add edges one by one
    addEdge(adj, 0, 1); // Directed edge from 0 to 1
    addEdge(adj, 0, 2); // Directed edge from 0 to 2
    addEdge(adj, 1, 2); // Directed edge from 1 to 2
    addEdge(adj, 2, 3); // Directed edge from 2 to 3

    cout << "Adjacency List Representation:" << endl;
    displayAdjList(adj);

    return 0;
}
```