**Graph (Day\_01)**

**Introduction of Graph:** Graph is non-linear data structure which consists of two things vertex/ node and edges

**Un Directed Graph:** In the undirected Graph the edges have no direction , So if the edge between node “a” and node "b”, so we can traverse in the both directions

**Directed Graph:** In directed Graph the edge have a direction, meaning if there is an edge from node “a” to node "b”, it does not imply that there is an edge from node "b” to node "a”

**Degree in Graph :** A number of edges are attached in a node in un-directed graph that will be a degree of that node

D = 2 D = 2

**D = 2**

**D=2 D=2**

**Total Number Degree =** D(a) + D(b) + D(c) + D(d) + D(e)

2 + 2 + 2 + 2 + 2 = 10

Total number of Degree = 10

## We can directly count the total number of degree using “**2\*E”**

Where **E** represents total number of Edges

Q. Why 2 in 2\*E??

Because every Edges is Associated with two nodes

**Directed Graph**: In directed Graph we have 2 things

1. In degree of particular node
2. Out Degree of particular node

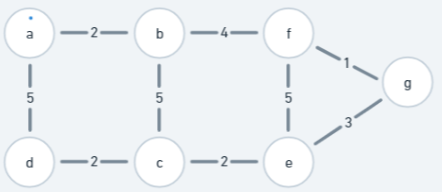
Out degree = 2 indegree = 1

in degree = 0 out-degree = 1

In Degree =2 indegree =1

Out-D = 0 out-degree = 1

**Edge Weights :**

****

If the weight is not define then the unite weight by default will be “1”

**Graph Representation:** we have two ways to store the graph

* **Matrix Way**
* **List Way**

**Matrix Representation: un directed Graph**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **node** | **0** | **1** | **2** | **3** | **4** | **5** |
| **0** | 0 | **0** | **0** | **0** | **0** | **0** |
| **1** | **0** | **0** | 1 | 1 | **0** | **0** |
| **2** | **0** | 1 | **0** | **0** | 1 | 1 |
| **3** | **0** | 1 | **0** | **0** | 1 | **0** |
| **4** | **0** | **0** | 1 | 1 | **0** | 1 |
| **5** | **0** | **0** | 1 | **0** | 1 | **0** |

**A diagram of a network

Description automatically generated**

**Edges connection:**

1 ---- 2, 2----1

1 ---- 3, 3 ----1

2 ---- 4, 4----2

2 ----5, 5 ----2

3 ---- 4, 4 ----3

4 ----5, 5 ----4

Sudo Code

Adj\_matrix∫(nodes, connection):

adj = Initialize the 2D matrix in (NxN)

for u , v 🡪connection:

adj[u][v] = 1

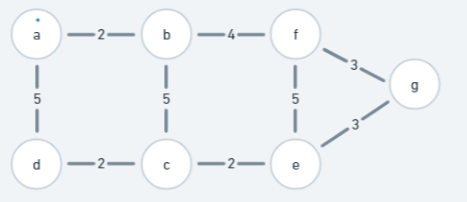
adj[v][u] = 1

return adj

Time Complexity: O(n x n)

Space Complexity: O(n x n)

**Matrix Representation:** Un -directed Graph Weighted Graph



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| node | a | b | c | d | e | f | g |
| a | 0 | 2 | 0 | 5 | 0 | 0 | 0 |
| b | 2 | 0 | 5 | 0 | 0 | 4 | 0 |
| c | 0 | 5 | 0 | 2 | 2 | 0 | 0 |
| d | 5 | 0 | 2 | 0 | 0 | 0 | 0 |
| e | 0 | 0 | 2 | 0 | 0 | 5 | 3 |
| f | 0 | 4 | 0 | 0 | 5 | 0 | 3 |
| g | 0 | 0 | 0 | 0 | 3 | 3 | 0 |

Edge Connection

a --- b, 2 c --- d, 2

a --- d, 5 e --- g, 3

b --- f , 4 f --- g, 3

b ---c, 5 e --- f, 5

c --- e, 2

matrix representation

Sudo Code

Adj\_matrix∫(nodes, connection):

adj = Initialize the 2D matrix in (NxN)

for u , v 🡪connection:

adj[u][v] = 1

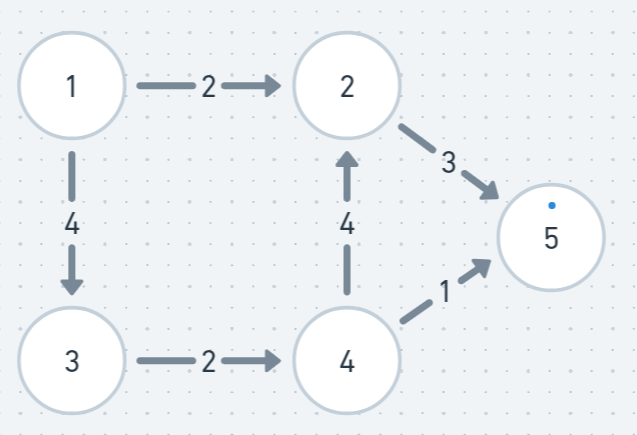
adj[v][u] = 1

return adj

Time Complexity: O(n x n)

Space Complexity: O(n x n)

**Matrix Representation:** directed Graph Weighted Graph

****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Node** | **1** | **2** | **3** | **4** | **5** |
| **1** | **0** | **2** | **4** | **0** | **0** |
| **2** | **0** | **0** | **0** | **0** | **3** |
| **3** | **0** | **0** | **0** | **2** | **0** |
| **4** | **0** | **4** | **0** | **0** | **1** |
| **5** | **0** | **0** | **0** | **0** | **0** |

**Edges Connection**

**1 🡪 2, 2**

**1 🡪3, 4**

**2 🡪 5, 3**

**3 🡪 4, 2**

**4 🡪 2, 4**

**4 🡪 5, 1**

Sudo Code for Directed Weighted Graph

Adj\_matrix∫(nodes, connection):

adj = Initialize the 2D matrix in (NxN)

for u , v , w🡪connection:

adj[u][v] = w

return adj

Time Complexity: O(n x n)

Space Complexity: O(n x n)

**2. List Representation (**for directed undirected and weighted and unweighted graph**)**

**2.1 undirected Graph**

**A diagram of a network

Description automatically generated**

1 – 2 0 – []

1 – 3 1 – [2,3]

2 – 4 2 – [1,4,5]

2 – 5 3 – [1,4]

3 – 4 4 – [2,3,5]

4 – 5 5 – [2, 4]

edges connection List Adjacency

Sudo Code

List\_adj ∫ (nodes, edges\_connection):

## we have to create a empty list of same size

Adj = [[] for \_ in range(nodes+1)]

for u, v in edges\_connection:

adj[u].append(v)

adj[v].append(u)

return adj

**Space Complexity:** O(nodes+2E)

**Time Complexity:** O(nodes+E)

* 1. **Un-Directed Weighted Graph** : List Adjacency

A diagram of a network

Description automatically generated

Edges weight List Adjacency

1 – 2 5 0 --- []

1 – 3 4 1 --- [(2,5),(3,4) ]

2 – 4 6 2 --- [(1,5),(4,6), (5,3)]

2 – 5 3 3 --- [(1,4),(4,6) ]

3 – 4 6 4 --- [(2,6),(3,6),(5,3)]

4 – 5 3 5 --- [(2,3),(4,3)]

**Sudo Code :**

Weighted\_undirected\_adj\_list**∫(**nodes, edges\_connection**):**

### Now let’s Definde the adj list

Adj = [[] for \_ in range(nodes +1)]

## Now let’s input the edges with weights

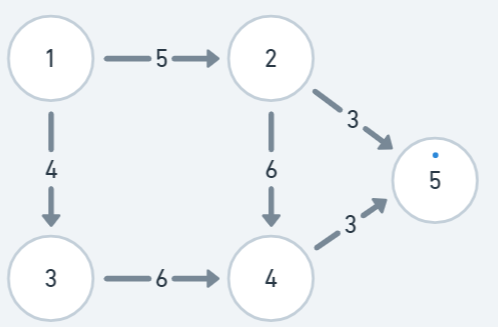
for u, v, w in edges\_connection:

adj[u].append((v,w)

adj[v].append((u,w))

return adj

**List Adjacency Directed Weighted Graph:**

****

Edges weight List Adjacency

1 – 2 5 0 --- []

1 – 3 4 1 --- [(2,5),(3,4) ]

2 – 4 6 2 --- [(4,6), (5,3)]

2 – 5 3 3 --- [(4,6) ]

3 – 4 6 4 --- [(5,3)]

4 – 5 3 5 --- []

**Sudo Code :**

Weighted\_undirected\_adj\_list**∫(**nodes, edges\_connection**):**

### Now let’s Definde the adj list

Adj = [[] for \_ in range(nodes +1)]

## Now let’s input the edges with weights

for u, v, w in edges\_connection:

adj[u].append((v,w)

return adj