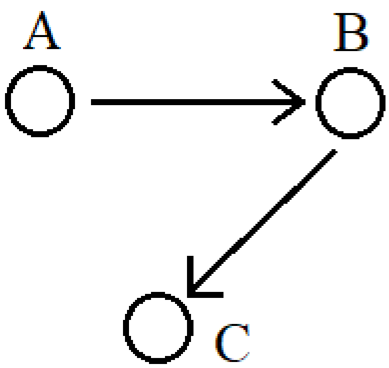
Dynamic Programming

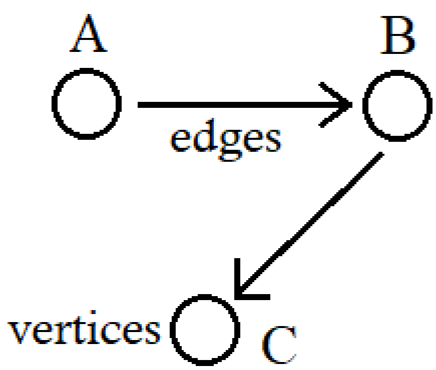
Topics:

- Graph – bfs & dfs & shortest paths.
- Backtrack
- Insertion Sort

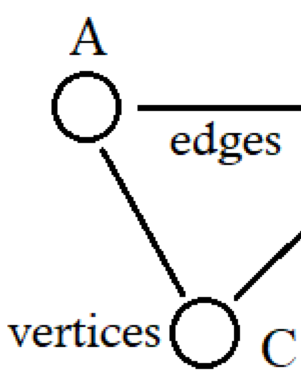
Graphs:

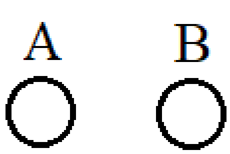
It is a data structure which has vertices and edges.





A graph can have cycles.



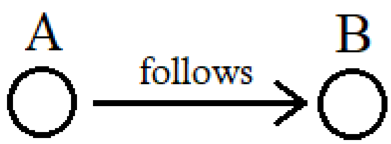


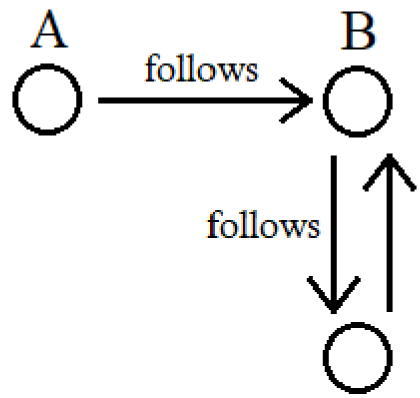
If we have two nodes and they are not connected even then it is called graph with zero edges.

Two types of Graphs:

1. Directed Graphs (it has direction)
2. Un-directed Graphs (it has no direction)

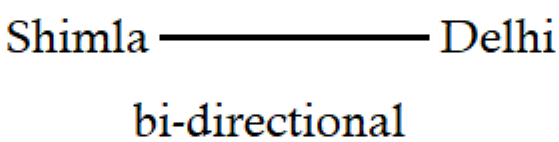
In social media platforms, A follows B and B follows C and C is following B. Such a type of graph is called **Directed Graphs**.

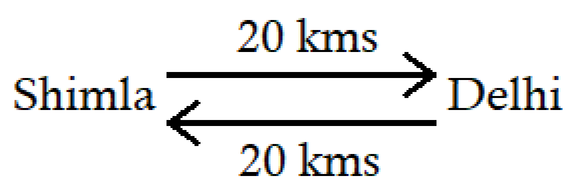




Directed Graphs are one **Direction Flow**.

When you don't put an arrow, then it means **Bi-Directional**.



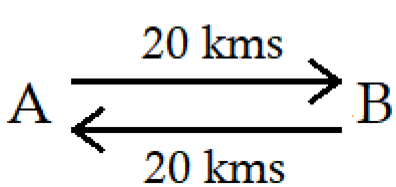


Bi-Directional

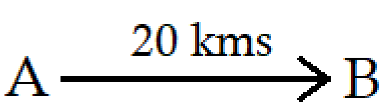
Un-directed Graph Directed Graph

Ex: Facebook, Google Maps etc

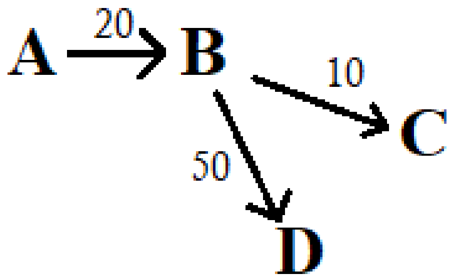
During back-end we will learn about graph data bases.



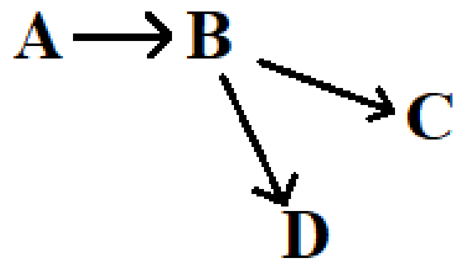
➔ An edge can have some weight for ex:



➔ It is called weighed directed graphs and if we don't have edge-weight, it is called un-weighted directed graph.



Weighted Directed Graph

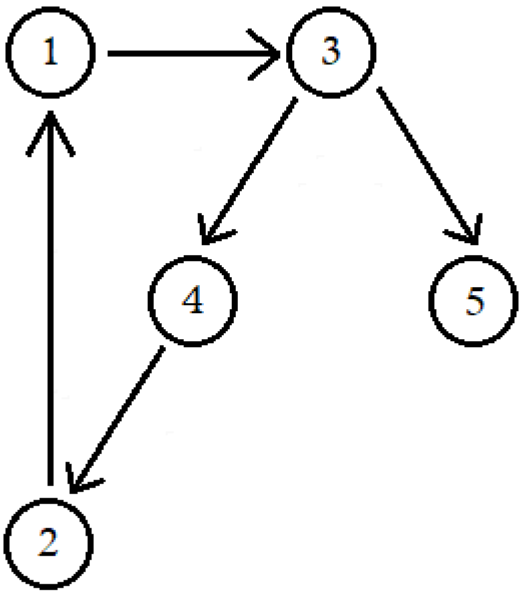


Un-Weighted Directed Graph.

Graph Construction:

2 approaches to construct graph

1) Adjacency Matrix
2) Adjacency List



$$\begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

To represent it in code, we need to create a matrix. The size of the matrix would be the number of vertices. If we have a zero in the matrix, then the vertices are not

$$\begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

connected.

Nothing in the graph is connected.

(1) → (3), is connected with an edge of 1

$$\begin{array}{cc} & \begin{array}{cccccc}0&1&2&3&4&5\end{array} \\ \begin{array}{c}0\\1\\2\\3\\4\\5\end{array} & \begin{bmatrix}0&0&0&0&0&0\\0&0&0&\boxed{1}&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\end{bmatrix}\end{array}$$

(3) → (4), is connected with an edge of 1

$$\begin{array}{cc} & \begin{array}{cccccc}0&1&2&3&4&5\end{array} \\ \begin{array}{c}0\\1\\2\\3\\4\\5\end{array} & \begin{bmatrix}0&0&0&0&0&0\\0&0&0&\boxed{1}&0&0\\0&0&0&0&0&0\\0&0&0&0&\boxed{1}&0\\0&0&0&0&0&0\\0&0&0&0&0&0\end{bmatrix}\end{array}$$

(3) → (5), is connected with an edge of 1

$$\begin{array}{cc} & \begin{array}{cccccc}0&1&2&3&4&5\end{array} \\ \begin{array}{c}0\\1\\2\\3\\4\\5\end{array} & \begin{bmatrix}0&0&0&0&0&0\\0&0&0&\boxed{1}&0&0\\0&0&0&0&0&0\\0&0&0&0&\boxed{1}&\boxed{1}\\0&0&0&0&0&0\\0&0&0&0&0&0\end{bmatrix}\end{array}$$

(4) → (2), is connected with an edge of 1

$$\begin{array}{cc} & \begin{array}{cccccc}0&1&2&3&4&5\end{array} \\ \begin{array}{c}0\\1\\2\\3\\4\\5\end{array} & \begin{bmatrix}0&0&0&0&0&0\\0&0&0&\boxed{1}&0&0\\0&0&0&0&0&0\\0&0&0&0&\boxed{1}&\boxed{1}\\0&0&\boxed{1}&0&0&0\\0&0&0&0&0&0\end{bmatrix}\end{array}$$

$$\begin{array}{cc} & \begin{array}{cccccc}0&1&2&3&4&5\end{array} \\ \begin{array}{c}0\\1\\2\\3\\4\\5\end{array} & \begin{bmatrix}0&0&0&0&0&0\\0&0&0&\boxed{1}&0&0\\0&\boxed{1}&0&0&0&0\\0&0&0&0&\boxed{1}&\boxed{1}\\0&0&\boxed{1}&0&0&0\\0&0&0&0&0&0\end{bmatrix}\end{array}$$

(2) → (1), is connected with an edge of 1

$$\begin{array}{ccc} & & \text{destination} \\ & & \begin{array}{cccccc}0&1&2&3&4&5\end{array} \\ \text{source} & \begin{array}{c}0\\1\\2\\3\\4\\5\end{array} & \begin{bmatrix}0&0&0&0&0&0\\0&0&0&\boxed{1}&0&0\\0&\boxed{1}&0&0&0&0\\0&0&0&0&\boxed{1}&\boxed{1}\\0&0&\boxed{1}&0&0&0\\0&0&0&0&0&0\end{bmatrix}\end{array}$$

These are mapped with source and destination.

To make a graph from Adjacence Matrix

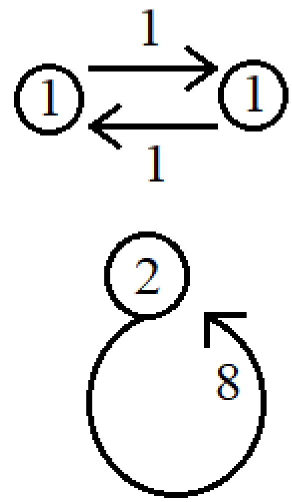
Ex:

$$\begin{array}{cc} & \begin{array}{ccc}0&1&2\end{array} \\ \begin{array}{c}0\\1\\2\end{array} & \begin{bmatrix}0&1&0\\1&0&0\\0&0&8\end{bmatrix}\end{array}$$

(0) → (1) are connected with length of 1

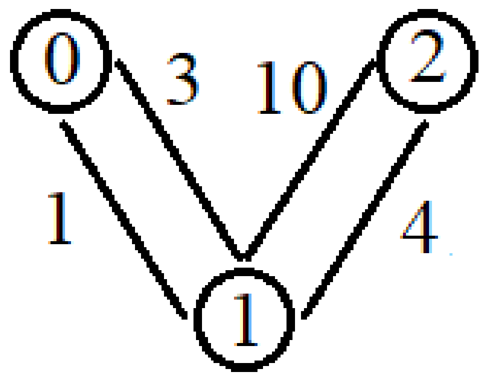
(1) → (0) are connected with length of 1

(2) → (2) are connected with edge of length 8

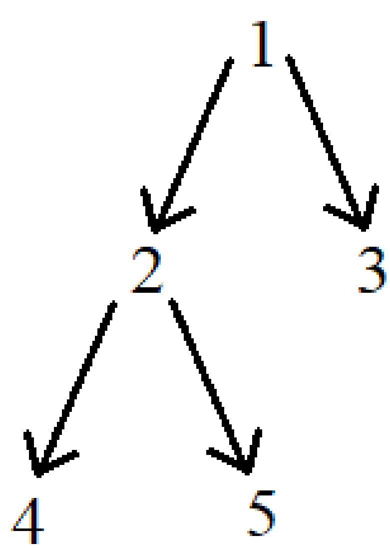


Ex:

$$\begin{array}{c c} & \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \end{array} & \begin{bmatrix} 0 & 1 & 0 \\ 3 & 0 & 4 \\ 0 & 10 & 0 \end{bmatrix} \end{array}$$

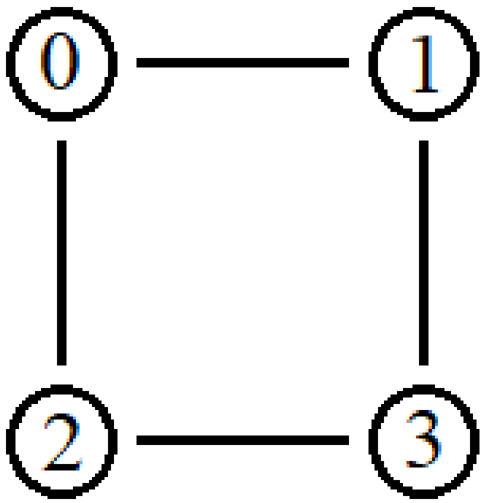


Can a tree be a graph? YES, because it has vertices and edges. It is a special type of graph that does not cycle. Here the number of edges = vertices – 1. So, for a tree,



$$\begin{array}{c c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

For,



$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & \\ 1 & \\ 2 & \\ 3 & \end{array}$$

$$\begin{array}{cc} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Adjacence Matrix for un-directed graph.

CODE:

```
n = 5
graph = [[0 for _ in range(n + 1)] for _ in range(n + 1)]

def addEdge(u, v, weight, directed):
    graph[u][v] = weight
    if not directed:
        graph[v][u] = weight

if __name__ == "__main__":
    addEdge(1, 2, 1, True)
    addEdge(0, 3, 1, True)
    addEdge(2, 3, 1, True)
    addEdge(3, 2, 1, True)
    addEdge(2, 5, 1, True)

    for i in range(n + 1):
        print(*graph[i])

# CAN ALSO BE WRITTEN AS:
    for i in range(n + 1):
        for j in range(n + 1):
            print(graph[i][j], end="")
        print()
```

OUTPUT:

0 0 0 1 0 0
0 0 1 0 0 0
0 0 0 1 0 1
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

For Un-Directed Graph,

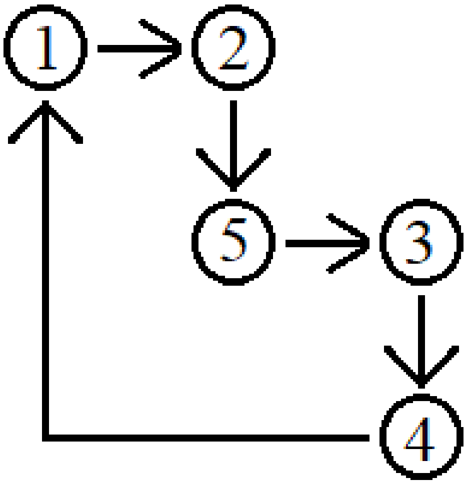
```
n = 5
graph = [[0 for _ in range(n + 1)] for _ in range(n + 1)]

def addEdge(u, v, weight, directed):
    graph[u][v] = weight
    if not directed:
        graph[v][u] = weight

if __name__ == "__main__":
    addEdge(1, 2, 1, False)
    addEdge(0, 3, 1, False)
    addEdge(2, 3, 1, False)
    addEdge(3, 2, 1, False)
    addEdge(2, 5, 1, False)

    for i in range(n + 1):
        for j in range(n + 1):
            print(graph[i][j], end="")
        print()
```

OUTPUT:

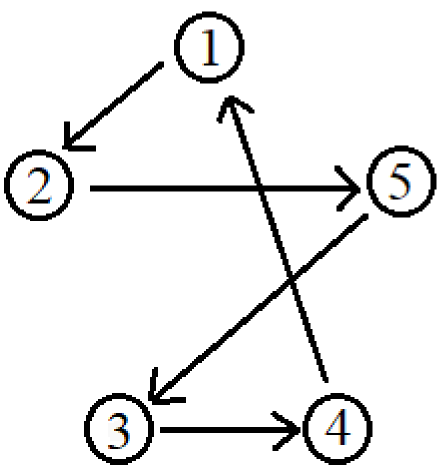
0 0 0 1 0 0
0 0 1 0 0 0
0 1 0 1 0 1
1 0 1 0 0 0
0 0 0 0 0 0
0 0 1 0 0 0

Adjacency List

In this, we will make a list. The indices are,



0 → [ ]
1 → [ ]
2 → [ ]
3 → [ ]
4 → [ ]
5 → [ ]



1 → 2, so add 2; 1 → [2]

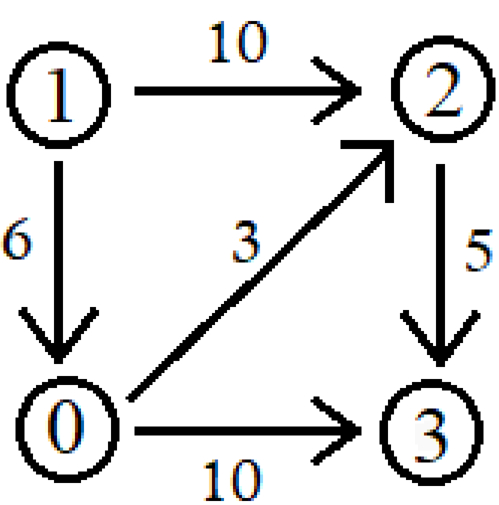
2 → 5, so add 5; 2 → [5]

3 → 4, so add 4; 3 → [4]

4 → 1, so add 1; 1 → [1]

5 → 4, so add 4; 5 → [3, 4]

Q)



Instead of adding vertices, we will be adding a tuple. So, the first value will be vertex and the second value will be weight (vertex, weight)

0 → [(3, 10), (2, 3)]

1 → [(0, 6), (2, 10)]

2 → [(3, 5)]

3 → [ ]

CODE:

```
graph = dict()

def addEdge(u, v, weight, directed):
    if u not in graph:
        graph[u] = list()

    graph[u].append((v, weight))


    if not directed:
        if v not in graph:
            graph[v] = list()
        graph[v].append((u, weight))

if __name__ == "__main__":
    addEdge(1, 2, 1, True)
    addEdge(0, 3, 1, True)
    addEdge(2, 3, 1, True)
    addEdge(3, 2, 1, True)
    addEdge(2, 5, 1, True)

    for key, value in graph.items():
        print(f"{key} has neighbour {value}")
```

OUTPUT:

1 has neighbour [(2, 1)]
0 has neighbour [(3, 1)]
2 has neighbour [(3, 1), (5, 1)]
3 has neighbour [(2, 1)]

Resources:

For Graphs – 2

- https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/