A **graph** is a data structure that consists of vertices that are connected via edges. It can be implemented with an:

1-adjacency matrix:

If we have a graph with N vertices, An adjacency matrix for the graph will be a N x N two-dimensional matrix. The rows and columns in the matrix represent the vertices of the graph and the values in the matrix determine whether there is an edge between two vertices or not. ..Suppose we have the adjacency matrix A for any graph. For any index (i,j), If there is an edge between vertex i and vertex j, we assign value 1 to A[i][j]. When there is no edge present between vertices i and j, The value 0 is assigned to A[i][j]. This can be implemented in Python as follows.

```
import numpy as np
 # keep vertices in a set
vertices = {0, 1, 2, 3, 4, 5}
# keep edges in a set
edges = {(0, 1), (1, 2), (0, 3), (1, 3), (3, 4), (2, 5), (4, 5), (2, 4)}
# create a 6X6 integer numpy array with all values initialised to zero
adjacencyMatrix = np.zeros((6, 6)).astype(int)
# Represent edges in the adjacency matrix
for edge in edges:
    v1 = edge[0]
    v2 = edge[1]
    adjacencyMatrix[v1][v2] = 1
    adjacencyMatrix[v2][v1] = 1 # if v1 is connected to v2, v2 is also connected to v1
print("The set of vertices of the graph is:")
print(vertices)
print("The set of edges of the graph is:")
print(edges)
print("The adjacency matrix representing the graph is:")
print(adjacencyMatrix)
```

Output:

```
The set of vertices of the graph is:
{0, 1, 2, 3, 4, 5}
The set of edges of the graph is:
{(0, 1), (2, 4), (1, 2), (3, 4), (0, 3), (4, 5), (2, 5), (1, 3)}
The adjacency matrix representing the graph is:
[[0 1 0 1 0 0]
 [1 0 1 1 0 0]
 [0 1 0 0 1 1]
 [1 1 0 0 1 0]
 [0 0 1 1 0 1]
 [0 0 1 0 1 0]]
```

Implementation of a graph using an adjacency matrix has a drawback. Here, we assign memory for each vertex whether it is present or not. This can be avoided by implementing the graph using the adjacency list as discussed in the following section.

2-**adjacency list**

An adjacency list stores a list of all connected vertices from each vertex. To implement this, we will use a dictionary in which each key of the dictionary represents a vertex and values for the keys contain a list of vertices the key vertex is connected to. This can be implemented as follows.

```
# keep vertices in a set
vertices = {0, 1, 2, 3, 4, 5}
# keep edges in a set
edges = {(0, 1), (1, 2), (0, 3), (1, 3), (3, 4), (2, 5), (4, 5), (2, 4)}
# create a dictionary with vertices of graph as keys and empty lists as values
adjacencyList={}
for vertex in vertices:
   adjacencyList[vertex]=[]
# Represent edges in the adjacency List
for edge in edges:
   v1 = edge[0]
   v2 = edge[1]
   adjacencyList[v1].append(v2)
   adjacencyList[v2].append(v1) # if v1 is connected to v2, v2 is also connected to v1
print("The set of vertices of the graph is:")
print(vertices)
print("The set of edges of the graph is:")
print(edges)
print("The adjacency List representing the graph is:")
print(adjacencyList)
```

Output:

```
The set of vertices of the graph is:
{0, 1, 2, 3, 4, 5}
The set of edges of the graph is:
{(0, 1), (2, 4), (1, 2), (3, 4), (0, 3), (4, 5), (2, 5), (1, 3)}
The adjacency List representing the graph is:
{0: [1, 3], 1: [0, 2, 3], 2: [4, 1, 5], 3: [4, 0, 1], 4: [2, 3, 5], 5: [4, 2]}
```

In the above output, we can see that each key is representing a vertex and every key is associated to a list of vertex it is connected to. This implementation is efficient than adjacency matrix representation of graph. This is because of the reason that we don’t need to store values for the edges which are not present.