**Task 4**

**Graph implementation in python**

A graph is a data structure that consists of vertices that are connected ​via edges. In an adjacency list representation of the graph, each vertex in the graph stores a list of neighboring vertices.

**1. Directed Graph Implementation**

Following is the Python implementation of a directed graph using an adjacency list:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | # A class to represent a graph object  class Graph:      # Constructor      def \_\_init\_\_(self, edges, n):          # allocate memory for the adjacency list          self.adjList = [[] for \_ in range(n)]            # add edges to the directed graph          for (src, dest) in edges:              # allocate node in adjacency list from src to dest              self.adjList[src].append(dest)    # Function to print adjacency list representation of a graph  def printGraph(graph):      for src in range(len(graph.adjList)):          # print current vertex and all its neighboring vertices          for dest in graph.adjList[src]:              print(f'({src} —> {dest}) ', end='')          print()    if \_\_name\_\_ == '\_\_main\_\_':        # Input: Edges in a directed graph      edges = [(0, 1), (1, 2), (2, 0), (2, 1), (3, 2), (4, 5), (5, 4)]        # No. of vertices (labelled from 0 to 5)      n = 6        # construct a graph from a given list of edges      graph = Graph(edges, n)        # print adjacency list representation of the graph      printGraph(graph) |

**2. Weighted Directed Graph Implementation**

In a weighted graph, every edge has a weight or cost associated with it.

Following is the Python implementation of a weighted directed graph using an adjacency list.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | # A class to represent a graph object  class Graph:      # Constructor to construct a graph      def \_\_init\_\_(self, edges, n):            # A list of lists to represent an adjacency list          self.adjList = [None] \* n            # allocate memory for the adjacency list          for i in range(n):              self.adjList[i] = []            # add edges to the directed graph          for (src, dest, weight) in edges:              # allocate node in adjacency list from src to dest              self.adjList[src].append((dest, weight))    # Function to print adjacency list representation of a graph  def printGraph(graph):      for src in range(len(graph.adjList)):          # print current vertex and all its neighboring vertices          for (dest, weight) in graph.adjList[src]:              print(f'({src} —> {dest}, {weight}) ', end='')          print()    if \_\_name\_\_ == '\_\_main\_\_':        # Input: Edges in a weighted digraph (as per the above diagram)      # Edge (x, y, w) represents an edge from `x` to `y` having weight `w`      edges = [(0, 1, 6), (1, 2, 7), (2, 0, 5), (2, 1, 4), (3, 2, 10),              (4, 5, 1), (5, 4, 3)]        # No. of vertices (labelled from 0 to 5)      n = 6        # construct a graph from a given list of edges      graph = Graph(edges, n)        # print adjacency list representation of the graph      printGraph(graph) |