class **Graph**(builtins.object)

| A class used to represent an undirected weighted graph

|

| *Attributes*

| ----------

| graph : Dict[object, List[object, int]]

| The adjacency list of vertices of the graph. For every vertex, its adjacent vertices are stored along with the edge weight.

| The weight must be a positive integer and it is equal to 1 for unweighted edges.

|

| *Methods*

| -------

| def addVertex(vert)

| addVertex(vert) adds a vertex to the graph.

|

| def addVerticesFromList(vertList)

| addVerticesFromList(vertList) adds a list of vertices to the graph.

|

| def addEdge(fromVert, toVert, weight = 1)

| addEdge(fromVert, toVert, weight) adds a new, weighted, undirected edge to the graph that connects two vertices.

| If not in the graph, the vertices should be added automatically.

|

| def addEdgesFromList(edgeList)

| addEdgesFromList(edgeList) adds a list of edges to the graph.

|

| def getVertices()

| getVertices() returns the list of all vertices in the graph.

|

| def getEdges()

| getEdges() returns the list of all edges in the graph.

|

| def getNeighbors(vertKey)

| getNeighbors(vertKey) returns the list of all neighbors of the vertex labeled vertKey.

|

| def vertIn(vertKey)

| in returns True for a statement of the form vertex in graph, if the given vertex is in the graph, False otherwise.

|

| def saveGraph(fileName, graphName)

| saveGraph(graph) writes dot representation of the graph to a text file

|

| def getShortestPaths(fromVert)

| getShortestPaths(fromVert) calculates shortest paths in the graph from the given vertex to all other vertices

|

| Methods defined here:

|

| **\_\_init\_\_(self)**

| Creates a new, empty undirected weighted graph.

|

| graph : Dict[object, List[object, int]]

|

| **addEdge(self, fromVert, toVert, weight=1)**

| Add a new, weighted, undirected edge to the graph that connects two vertices

| if edge is not already in the graph. If these vertices are not in the graph, they

| add automatically.

|

| If the argument `weight` isn't passed in, the default weight is used.

|

| *def addEdge(self, fromVert, toVert, weight = 1) -> NoReturn*

|

| *Parameters*

| ----------

| fromVert : object

| The first vertex connected to the edge we want to add to the graph

| toVert : object

| The second vertex connected to the edge we want to add to the graph

| weight : object

| The weight of the edge (default is 1 for an unweighted edge). Must be a positive integer

|

| **addEdgesFromList(self, edgeList)**

| Add a list of edges to the graph if they are not already in the graph.

|

| *def addEdgesFromList(self, edgeList: List[object]) -> NoReturn*

|

| *Parameters*

| ----------

| edgeList : List[object]

| A list of edges we want to add to the graph

|

| **addVertex(self, vert)**

| Adds a vertex to the graph if it is not already in the graph.

|

| *def addVertex(self, vert: object) -> NoReturn*

|

| *Parameters*

| ----------

| vert : object

| The vertex we want to add to the graph

|

| **addVerticesFromList(self, vertList)**

| Add a list of vertices to the graph if they are not already in the graph.

|

| *def addVerticesFromList(self, vertList: List[object]) -> NoReturn*

|

| *Parameters*

| ----------

| vertList : List[object]

| A list of vertices we want to add to the graph

|

| **getEdges(self)**

| Return the list of all edges in the graph.

|

| *def getEdges(self) -> List[List[object, int]]*

|

| *Returns*

| -------

| list

| A list of all edges in the graph. It includes vertices connected to the edge and its weight

|

| **getNeighbors(self, vertKey)**

| Returns the list of all neighbors of the vertex labeled vertKey, if the vertex is in the graph.

|

| *def getNeighbors(self, vertKey: object) -> List[List[object, int]]*

|

| *Parameters*

| ----------

| vertKey : object

| The vertex whose neighbors we want to know

|

| *Returns*

| -------

| list

| a list of all neighbors of the vertex labeled vertKey

|

| **getShortestPaths(self, fromVert)**

| Calculates shortest path in the graph from the given vertex to all other vertices, if the vertex is in the graph.

| Use Dijkstra’s algorithm. Dijkstra’s algorithm maintains a set S of vertices whose final shortest-path weights from the source

| s have already been determined. The algorithm repeatedly selects the vertex u from V-S with the minimum shortest-path estimate,

| adds u to S, and relaxes all edges leaving u. We use a min-priority queue q of vertices, keyed by their d values.

|

| *def getShortestPaths(self, fromVert: object) -> List[Tuple[str, List[int, str]]]*

|

| *Parameters*

| ----------

| fromVert : object

| The vertex whose the shortest paths we want to calculate

|

| *Returns*

| -------

| list

| a list of the shortest paths. Have the neighbor's name, the length of the shortest path, and vertices included in this path

|

| **getVertices(self)**

| Return the list of all vertices in the graph.

|

| *def addVertex(self) -> List[object]*

|

| *Returns*

| -------

| list

| A list of all vertices in the graph

|

| **saveGraph(self, fileName, graphName)**

| Write dot representation of the graph to a text file. Create a new text file with

| dot representation of the graph in the same directory where the script file locates.

|

| *def saveGraph(self, fileName: str, graphName: str) -> NoReturn*

|

| *Parameters*

| ----------

| fileName : str

| Text file name

| graphName : str

| Graph name in the text file

|

| **vertIn(self, vertKey)**

| Return True for a statement of the form vertex in graph, if the given vertex is in the graph, False otherwise.

|

| *def vertIn(self, vertKey: object) -> bool*

|

| *Parameters*

| ----------

| vertKey : object

| The vertex we want to check

|

| *Returns*

| -------

| bool

| Result of checking if the vertex is in the graph

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)