***The function from the class Graph***

def bellman\_ford(self, start\_vertex):  
 *"""  
 Time Complexity: O(number\_of\_vertices \* number\_of\_edges)  
 Bellman-Ford algorithm extended to find the shortest path from a source vertex to all other vertices in the graph,  
 track the path, and detect negative weight cycles that are reachable from the source.  
  
 :param start\_vertex: The source vertex  
 :return: Tuple containing three elements:  
 - A matrix of distances from the source to each vertex.  
 - A matrix with the path from the source to each vertex.  
 - A boolean indicating whether a negative cycle was detected.  
 """* # Initialize distances and predecessors  
 distance = [[float('inf')] \* (self.\_\_number\_of\_vertices + 1) for \_ in range(self.\_\_number\_of\_vertices)]  
 predecessor = [[-1] \* (self.\_\_number\_of\_vertices + 1) for \_ in range(self.\_\_number\_of\_vertices)]  
 distance[start\_vertex][0] = 0  
  
 for k in range(1, self.\_\_number\_of\_vertices + 1):  
 for v in range(self.\_\_number\_of\_vertices):  
 distance[v][k] = distance[v][k - 1]  
 predecessor[v][k] = predecessor[v][k - 1]  
  
 for u, v in self.parse\_cost():  
 if distance[u][k - 1] + self.costs[(u, v)] < distance[v][k]:  
 distance[v][k] = distance[u][k - 1] + self.costs[(u, v)]  
 predecessor[v][k] = u  
  
 # Check for negative weight cycles  
 for v in range(self.\_\_number\_of\_vertices):  
 if distance[v][self.\_\_number\_of\_vertices - 1] < distance[v][self.\_\_number\_of\_vertices]:  
 return distance, predecessor, True  
  
 return distance, predecessor, False

***The function from the UI***

def find\_lowest\_cost\_path(self):  
 *"""  
 A function that finds in the current graph the path with the smallest cost between two given vertices.  
 :return: None, but it prints the cost and the corresponding path or a message if there is no such path  
 """* start\_vertex = int(input("Enter the start vertex: "))  
 goal\_vertex = int(input("Enter the goal vertex: "))  
 distance, predecessor, has\_negative\_cycle = self.\_\_graphs[self.\_\_current].bellman\_ford(start\_vertex)  
  
 if has\_negative\_cycle:  
 print("Negative cost cycle detected, reachable from vertex {}!".format(start\_vertex))  
 else:  
 n = len(distance[0]) - 1  
 minimum\_cost = float('inf')  
 best\_k = -1  
 for k in range(n + 1):  
 if distance[goal\_vertex][k] < minimum\_cost:  
 minimum\_cost = distance[goal\_vertex][k]  
 best\_k = k  
  
 if minimum\_cost == float('inf'):  
 print(f"Vertex {goal\_vertex} is unreachable from {start\_vertex}.")  
 return  
  
 path = []  
 current = goal\_vertex  
 k = best\_k  
 while current is not None and k >= 0:  
 path.insert(0, current)  
 current = predecessor[current][k]  
 k -= 1  
 print(f"The path is {path} and the cost is {minimum\_cost}.\n")