def hamiltonian\_cycle\_heuristic(self):  
 # sort edges. We save the edges aas a list of type: [[[vertex1,vertex2], cost], [[vertex3,vertex4], cost]] ....  
 edges = [[[key[0], key[1]], value] for key, value in sorted(self.\_dictCost.items(), key=lambda item: item[1])]  
 visited = []  
 path = []  
 degree = [0] \* self.get\_number\_vertices()  
 cost = 0  
  
 # start algorithm  
 nr\_edges\_left = self.get\_number\_vertices()  
 for edge in edges:  
 vertex1 = edge[0][0]  
 vertex2 = edge[0][1]  
 if degree[vertex1]+1 <= 2 and degree[vertex2]+1 <= 2: # can't visit more than twice  
 if vertex1 in visited and vertex2 in visited: # cycle ends (or algorithm)  
 if nr\_edges\_left == 1:  
 nr\_edges\_left = 0  
 path.append(edge[0])  
 cost += edge[1]  
 break  
 continue # else it ends too soon so we go to next edge  
 degree[vertex1] += 1  
 degree[vertex2] += 1  
 remaining\_vertex = [v for v in [vertex1, vertex2] if v not in visited]  
  
 if len(remaining\_vertex) == 2:  
 visited.append(remaining\_vertex[0])  
 visited.append(remaining\_vertex[1])  
 else:  
 visited.append(remaining\_vertex)  
  
 path.append(edge[0])  
 nr\_edges\_left -= 1  
 cost += edge[1]  
  
 if nr\_edges\_left != 0:  
 print("No cycle!")  
 else:  
 path\_vertices = list()  
 # add to the path the vertices from the first edge then remove the edge  
 path\_vertices.append(path[0][0])  
 path\_vertices.append(path[0][1])  
 path.remove(path[0])  
 nr\_edges\_left = self.get\_number\_vertices() - 1  
  
 while nr\_edges\_left:  
 search\_vertex = path\_vertices[len(path\_vertices)-1] # get last vertex  
 # search for edge which contains the last vertex, then get the other vertex from the founded edge  
 connected\_edge = [edge for edge in path if edge[1] == search\_vertex or edge[0] == search\_vertex]  
 remaining\_vertex = [v for v in connected\_edge[0] if v != search\_vertex]  
 # add the other vertex to the path of vertices and remove the founded edge from the list  
 path\_vertices.append(remaining\_vertex[0])  
 path.remove(connected\_edge[0])  
 nr\_edges\_left -= 1  
 print(f"The cycle is: {path\_vertices} and has the cost {cost}")

def hamiltonian\_cycle\_greedy(self):  
 n = self.get\_number\_vertices()  
 optimal\_cost = 0  
 optimal\_path = []  
 vertex = 0  
 optimal\_path.append(vertex)  
 visited = [0] \* n  
 visited[vertex] = 1  
 nr\_edges = self.get\_number\_vertices()  
 for \_ in range(n):  
 min\_cost, vertex\_out = self.choose(vertex, visited)  
  
 if vertex\_out != -1:  
 visited[vertex\_out] = 1  
 optimal\_path.append(vertex\_out)  
 optimal\_cost += min\_cost  
 nr\_edges -= 1  
 vertex = vertex\_out  
 else:  
 break  
  
 if not self.is\_edge(optimal\_path[0], optimal\_path[len(optimal\_path)-2]) or nr\_edges > 1:  
 print("No cycle!")  
 else:  
 optimal\_cost += self.get\_cost(optimal\_path[0], optimal\_path[len(optimal\_path)-1])  
 optimal\_path.append(optimal\_path[0])  
 print(f"The cycle is: {optimal\_path} and has the cost {optimal\_cost}")