Documentation Practical Work no. 5

Code added(Brute force method)

Repository

```
def calculate_cycle_cost(self, cycle: list) -> int:
  '''
  This function calculates the cost of a cycle in a graph, provided that the graph is complete.
  So, there will be no need to check if the edge exists.
  Also, the cycle will always be valid.
  The function returns the cost of the cycle.
  '''
  cost = 0
  n = len(cycle)
  for i in range(n):
      cost += self.graph.get_cost(cycle[i], cycle[(i + 1) % n])
  return cost

def TSP_brute_force(self) -> tuple:
  '''
  Travelling salesman problem - brute force method
  This function solves the TSP using a brute force method
  The function returns a tuple consisting of the minimum cost and the cycle which
  generated that mininum cost
  '''
  if (self.number_of_edges() != self.number_of_vertices() * (self.number_of_vertices() - 1)):
      raise RepoError("\nGraph is not complete\n")
  vertices = self.get_vertices()
  min_cycle = []
  min_cost = float('inf')
  for permutation in itertools.permutations(vertices):
      current_cost = self.calculate_cycle_cost(permutation)
      if current_cost < min_cost:
          min_cost = current_cost
          min_cycle = permutation
  return min_cost, min_cycle
```

Service

```
def TSP_brute_force(self) -> tuple:
    '''
    Returns the shortest path in the Travelling Salesman Problem using brute force
    Args:
        start : the starting vertex
        target : the ending vertex
    Preconditions:
        The graph is complete (symetric or asymetric)
    '''
    return self.repo.TSP_brute_force()
```

UI

```
if command == "1":
    min_cost, min_cycle = self.service.TSP_brute_force()
    print(f"\nMinimum cost: {min_cost}")
    print("Path: ", end="")
    for vertex in min_cycle:
        print(vertex, end=" -> ")
    print(min_cycle[0])
```

Bonus 1(Greedy method)

Repository

```
def TSP_greedy(self) -> tuple:
  '''
  Travelling salesman problem - greedy method
  This function solves the TSP using a greedy method
  The function returns a tuple consisting of the minimum cost and the cycle which
  NOTE: that the greedy method does not always return the optimal solution!
  It just returns a solution which is close to the optimal one in a much smaller time frame
  '''
  n = self.number_of_vertices()
  final_cost = float('inf')
  final_cycle = []
  for start in range(n):
      # takes into consideration all possible starting vertices
      # to get a better approximation
      visited = [False] * n
      cycle = [start]
      visited[start] = True
      total_cost = 0
      current = start
      for _ in range(n - 1):
          # to ensure that we visit all other vertices exactly once
          next_vertex = None
          min_cost = float('inf')
          for v in range(n):
              if not visited[v] and self.graph.get_cost(current, v) < min_cost:
                  next_vertex = v
```

```
                min_cost = self.graph.get_cost(current, v)
        cycle.append(next_vertex)
        visited[next_vertex] = True
        total_cost += min_cost
        current_vertex = next_vertex
    cycle.append(start)
    if total_cost < final_cost:
        final_cost = total_cost
        final_cycle = cycle
return final_cost, final_cycle
```

Service

```
def TSP_greedy(self) -> tuple:
  '''
  Returns the shortest path in the Travelling Salesman Problem using greedy algorithm
  Args:
      start : the starting vertex
      target : the ending vertex
  Preconditions:
      The graph is complete (symetric or asymetric)
  '''
  return self.repo.TSP_greedy()
```

UI

```
elif command == "2":
  min_cost, min_cycle = self.service.TSP_greedy()
  print(f"\nMinimum cost: {min_cost}")
  print("Path: ", end="")
  for vertex in min_cycle[:-1]:
      print(vertex, end=" -> ")
  print(min_cycle[-1])
```