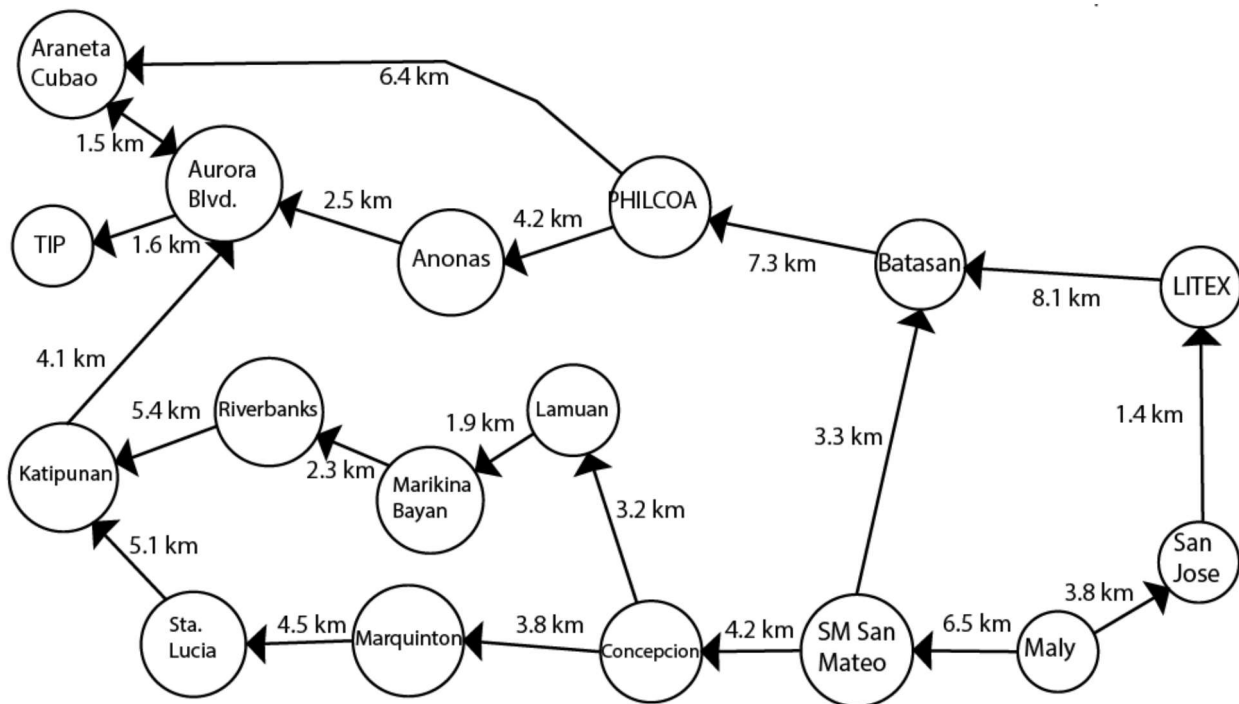


## CASE STUDY 1

- Choose one (1) real-world problem. Discuss the given scenario or the problem.
- Create an algorithm to solve the given problem.
- Apply applicable techniques to solve the problems (optimization, dynamic programming or graph)
- Create a slide presentation of the case study.
- Create a folder in your Github repo for the course; the folder must contain your source code and pdf of the presentation.

### ✓ PROBLEM

A student of TIP who lives in Maly San Mateo is having a hard time deciding which route he will take in order for him to save money and time. With the given Graph below, we can see the distance between the landmarks that he will come across during his trip to TIP.



The estimated fare for the public transportation is also listed below.

## ROUTE 1

Maly to Marikina Bayan: 25 PHP

Marikina Bayan to TIP: 15 PHP

## ROUTE 2

Maly to PHILCOA: 25 PHP

PHILCOA to Araneta Cubao: 15 PHP

Araneta Cubao to TIP: 11 PHP

## ROUTE 3

Maly to Sta Lucia: 35 PHP

Sta Lucia to TIP: 20 PHP

## ROUTE 4

Maly to PHILCOA: 25 PHP

PHILCOA to Anonas: 13 PHP

## ROUTE 5

Maly to San Jose: 12 PHP

San Jose to Cubao: 50 PHP

The code created aims to provide the student the shortest route he can take from his house Maly, to hist destination TIP

```

1 class Node(object):
2     def __init__(self, name):
3         """Assumes name is a string"""
4         self.name = name
5
6     def getName(self):
7         return self.name
8
9     def __str__(self):
10        return self.name
11
12 class Edge(object):
13     def __init__(self, src, dest, distance):
14         """Assumes src and dest are nodes"""
15         self.src = src
16         self.dest = dest
17         self.distance = distance
18     def getDistance(self):
19         return self.distance
20     def getSource(self):
21         return self.src
22     def getDestination(self):
23         return self.dest
24     def __str__(self):
25         return f"{self.src.getName()} -> {self.dest.getName()} Distance: {self.distance} km"
26
27 class Digraph(object):
28     """edges is a dict mapping each node to a list of
29     its children"""
30     def __init__(self):
31         self.edges = {}
32     def addNode(self, node):
33         if node in self.edges:
34             raise ValueError('Duplicate node')
35         else:
36             self.edges[node] = []
37     def addEdge(self, edge):
38         src = edge.getSource()
39         dest = edge.getDestination()
40         if not (src in self.edges and dest in self.edges):
41             raise ValueError('Node not in graph')
42         self.edges[src].append(dest)
43     def childrenOf(self, node):
44         return self.edges[node]
45     def hasNode(self, node):
46         return node in self.edges
47     def getNode(self, name):
48         for n in self.edges:
49             if n.getName() == name:
50                 return n
51         raise NameError(name)
52
53     def __str__(self):
54         result = ''
55         for src in self.edges:
56             for dest, distance in self.edges[src]:
57                 result += f"{src.getName()} -> {dest.getName()} Distance: {distance} km\n"
58         return result[:-1]
59
60 class Graph(Digraph):
61     def addEdge(self, edge):
62         Digraph.addEdge(self, edge)
63         rev = Edge(edge.getDestination(), edge.getSource(), edge.getDistance())
64         Digraph.addEdge(self, rev)
65
66 def buildGraph(graphType):
67     g = graphType()
68     for name in ('Maly', 'San Jose', 'LITEX', 'Batasan', 'Philcoa',
69                 'Kalayaan Ave.', 'Araneta-Cubao', 'Aurora Blvd.',
70                 'TIP', 'Anonas', 'SM San Mateo', 'Concepcion',
71                 'Lamuan', 'Marikina-Bayan', 'Marquinton', 'Sta. Lucia',
72                 'Katipunan Ave.', 'Riverbanks'):
73         g.addNode(Node(name))
74
75     g.addEdge(Edge(g.getNode('Maly'), g.getNode('San Jose'), 3.8))
76     g.addEdge(Edge(g.getNode('San Jose'), g.getNode('LITEX'), 1.4))
77     g.addEdge(Edge(g.getNode('LITEX'), g.getNode('Batasan'), 8.1))

```

```

78 g.addEdge(Edge(g.getNode('Batasan'), g.getNode('Philcoa'),7.3))
79 g.addEdge(Edge(g.getNode('Philcoa'), g.getNode('Araneta-Cubao'),6.4))
80 g.addEdge(Edge(g.getNode('Araneta-Cubao'), g.getNode('Aurora Blvd. '),1.5))
81 g.addEdge(Edge(g.getNode('Aurora Blvd. '), g.getNode('TIP'),1.6))
82 g.addEdge(Edge(g.getNode('Philcoa'), g.getNode('Anonas'),4.2))
83 g.addEdge(Edge(g.getNode('Anonas'), g.getNode('Aurora Blvd. '),2.5))
84 g.addEdge(Edge(g.getNode('Maly'), g.getNode('SM San Mateo'),6.5))
85 g.addEdge(Edge(g.getNode('SM San Mateo'), g.getNode('Batasan'),3.3))
86 g.addEdge(Edge(g.getNode('SM San Mateo'), g.getNode('Concepcion'),4.2))
87 g.addEdge(Edge(g.getNode('Concepcion'), g.getNode('Lamuan'),3.2))
88 g.addEdge(Edge(g.getNode('Concepcion'), g.getNode('Marquinton'),3.8))
89 g.addEdge(Edge(g.getNode('Lamuan'), g.getNode('Marikina-Bayan'),1.9))
90 g.addEdge(Edge(g.getNode('Marikina-Bayan'), g.getNode('Riverbanks'),2.3))
91 g.addEdge(Edge(g.getNode('Riverbanks'), g.getNode('Katipunan Ave. '),5.4))
92 g.addEdge(Edge(g.getNode('Marquinton'), g.getNode('Sta. Lucia'),4.5))
93 g.addEdge(Edge(g.getNode('Sta. Lucia'), g.getNode('Katipunan Ave. '),5.1))
94 g.addEdge(Edge(g.getNode('Katipunan Ave. '), g.getNode('Aurora Blvd. '),4.1))
95
96 return g
97
98 def printPath(path):
99     """Assumes path is a list of nodes"""
100     result = ''
101     for i in range(len(path)):
102         result = result + str(path[i])
103         if i != len(path) - 1:
104             result = result + '->'
105     return result
106
107
108 def DFS(graph, start, end, path, shortest, toPrint = False):
109     path = path + [start]
110     if toPrint:
111         print('Current DFS path:', printPath(path))
112     if start == end:
113         return path
114     for node in graph.childrenOf(start):
115         if node not in path:
116             if shortest == None or len(path) < len(shortest):
117                 newPath = DFS(graph, node, end, path, shortest,
118                             toPrint)
119                 if newPath != None:
120                     shortest = newPath
121             elif toPrint:
122                 print('Already visited', node)
123     return shortest
124
125
126 def shortestPath(graph, start, end ,toPrint = False):
127     """Assumes graph is a Digraph; start and end are nodes
128     Returns a shortest path from start to end in graph"""
129     return DFS(graph, start, end, [], None, toPrint)
130
131 def testSP(source, destination):
132     g = buildGraph(Digraph)
133     sp = shortestPath(g, g.getNode(source), g.getNode(destination),
134                     toPrint = True)
135     if sp != None:
136         print('Shortest path from', source, 'to',
137             destination, 'is', printPath(sp))
138     else:
139         print('There is no path from', source, 'to', destination)
140
141 testSP('Maly', 'TIP')
142
143

```

```

Current DFS path: Maly
Current DFS path: Maly->San Jose
Current DFS path: Maly->San Jose->LITEX
Current DFS path: Maly->San Jose->LITEX->Batasan
Current DFS path: Maly->San Jose->LITEX->Batasan->Philcoa
Current DFS path: Maly->San Jose->LITEX->Batasan->Philcoa->Araneta-Cubao
Current DFS path: Maly->San Jose->LITEX->Batasan->Philcoa->Araneta-Cubao->Aurora Blvd.
Current DFS path: Maly->San Jose->LITEX->Batasan->Philcoa->Araneta-Cubao->Aurora Blvd.->TIP
Current DFS path: Maly->San Jose->LITEX->Batasan->Philcoa->Anonas
Current DFS path: Maly->San Jose->LITEX->Batasan->Philcoa->Anonas->Aurora Blvd.

```

```

Current DFS path: Maly->San Jose->LITEX->Batasan->Philcoa->Anonas->Aurora Blvd.->TIP
Current DFS path: Maly->SM San Mateo
Current DFS path: Maly->SM San Mateo->Batasan
Current DFS path: Maly->SM San Mateo->Batasan->Philcoa
Current DFS path: Maly->SM San Mateo->Batasan->Philcoa->Araneta-Cubao
Current DFS path: Maly->SM San Mateo->Batasan->Philcoa->Araneta-Cubao->Aurora Blvd.
Current DFS path: Maly->SM San Mateo->Batasan->Philcoa->Araneta-Cubao->Aurora Blvd.->TIP
Current DFS path: Maly->SM San Mateo->Batasan->Philcoa->Anonas
Current DFS path: Maly->SM San Mateo->Batasan->Philcoa->Anonas->Aurora Blvd.
Current DFS path: Maly->SM San Mateo->Batasan->Philcoa->Anonas->Aurora Blvd.->TIP
Current DFS path: Maly->SM San Mateo->Concepcion
Current DFS path: Maly->SM San Mateo->Concepcion->Lamuan
Current DFS path: Maly->SM San Mateo->Concepcion->Lamuan->Marikina-Bayan
Current DFS path: Maly->SM San Mateo->Concepcion->Lamuan->Marikina-Bayan->Riverbanks
Current DFS path: Maly->SM San Mateo->Concepcion->Lamuan->Marikina-Bayan->Riverbanks->Katipunan Ave.
Current DFS path: Maly->SM San Mateo->Concepcion->Marquinton
Current DFS path: Maly->SM San Mateo->Concepcion->Marquinton->Sta. Lucia
Current DFS path: Maly->SM San Mateo->Concepcion->Marquinton->Sta. Lucia->Katipunan Ave.
Current DFS path: Maly->SM San Mateo->Concepcion->Marquinton->Sta. Lucia->Katipunan Ave.->Aurora Blvd.
Shortest path from Maly to TIP is Maly->SM San Mateo->Batasan->Philcoa->Anonas->Aurora Blvd.->TIP

```

```

1 class Node(object):
2     def __init__(self, name):
3         self.name = name
4
5     def getName(self):
6         return self.name
7
8     def __str__(self):
9         return self.name
10
11 class Edge(object):
12     def __init__(self, src, dest, distance):
13         self.src = src
14         self.dest = dest
15         self.distance = distance
16
17     def getDistance(self):
18         return self.distance
19
20     def getSource(self):
21         return self.src
22
23     def getDestination(self):
24         return self.dest
25
26     def __str__(self):
27         return f"{self.src.getName()} -> {self.dest.getName()} Distance: {self.distance} km"
28
29 class Digraph(object):
30     def __init__(self):
31         self.edges = {}
32
33     def addNode(self, node):
34         if node in self.edges:
35             raise ValueError('Duplicate node')
36         else:
37             self.edges[node] = []
38
39     def addEdge(self, edge):
40         src = edge.getSource()
41         dest = edge.getDestination()
42         if not (src in self.edges and dest in self.edges):
43             raise ValueError('Node not in graph')
44         self.edges[src].append((dest, edge.getDistance()))
45
46     def childrenOf(self, node):
47         return self.edges[node]
48
49     def hasNode(self, node):
50         return node in self.edges
51
52     def getNode(self, name):
53         for n in self.edges:
54             if n.getName() == name:
55                 return n
56         raise NameError(name)
57
58     def __str__(self):

```

```

50     def __str__(self):
51         result = ''
52         for src in self.edges:
53             for dest, distance in self.edges[src]:
54                 result += f"{src.getName()} -> {dest.getName()} Distance: {distance} km\n"
55         return result[:-1]
56
57 class Graph(Digraph):
58     def addEdge(self, edge):
59         Digraph.addEdge(self, edge)
60         rev = Edge(edge.getDestination(), edge.getSource(), edge.getDistance())
61         Digraph.addEdge(self, rev)
62
63 def buildGraph(graphType):
64     g = graphType()
65     for name in ('Maly', 'San Jose', 'LITEX', 'Batasan', 'Philcoa',
66                 'Kalayaan Ave.', 'Araneta-Cubao', 'Aurora Blvd.',
67                 'TIP', 'Anonas', 'SM San Mateo', 'Concepcion',
68                 'Lamuan', 'Marikina-Bayan', 'Marquinton', 'Sta. Lucia',
69                 'Katipunan Ave.', 'Riverbanks'):
70         g.addNode(Node(name))
71
72     g.addEdge(Edge(g.getNode('Maly'), g.getNode('San Jose'), 3.8))
73     g.addEdge(Edge(g.getNode('San Jose'), g.getNode('LITEX'), 1.4))
74     g.addEdge(Edge(g.getNode('LITEX'), g.getNode('Batasan'), 8.1))
75     g.addEdge(Edge(g.getNode('Batasan'), g.getNode('Philcoa'), 7.3))
76     g.addEdge(Edge(g.getNode('Philcoa'), g.getNode('Araneta-Cubao'), 6.4))
77     g.addEdge(Edge(g.getNode('Araneta-Cubao'), g.getNode('Aurora Blvd.'), 1.5))
78     g.addEdge(Edge(g.getNode('Aurora Blvd.'), g.getNode('TIP'), 1.6))
79     g.addEdge(Edge(g.getNode('Philcoa'), g.getNode('Anonas'), 4.2))
80     g.addEdge(Edge(g.getNode('Anonas'), g.getNode('Aurora Blvd.'), 2.5))
81     g.addEdge(Edge(g.getNode('Maly'), g.getNode('SM San Mateo'), 6.5))
82     g.addEdge(Edge(g.getNode('SM San Mateo'), g.getNode('Batasan'), 3.3))
83     g.addEdge(Edge(g.getNode('SM San Mateo'), g.getNode('Concepcion'), 4.2))
84     g.addEdge(Edge(g.getNode('Concepcion'), g.getNode('Lamuan'), 3.2))
85     g.addEdge(Edge(g.getNode('Concepcion'), g.getNode('Marquinton'), 3.8))
86     g.addEdge(Edge(g.getNode('Lamuan'), g.getNode('Marikina-Bayan'), 1.9))
87     g.addEdge(Edge(g.getNode('Marikina-Bayan'), g.getNode('Riverbanks'), 2.3))
88     g.addEdge(Edge(g.getNode('Riverbanks'), g.getNode('Katipunan Ave.'), 5.4))
89     g.addEdge(Edge(g.getNode('Marquinton'), g.getNode('Sta. Lucia'), 4.5))
90     g.addEdge(Edge(g.getNode('Sta. Lucia'), g.getNode('Katipunan Ave.'), 5.1))
91     g.addEdge(Edge(g.getNode('Katipunan Ave.'), g.getNode('Aurora Blvd.'), 4.1))
92
93     return g
94
95 def printPath(path):
96     result = ''
97     for i in range(len(path)):
98         result = result + str(path[i])
99         if i != len(path) - 1:
100             result = result + '->'
101     return result
102
103 def calculateDistance(graph, path):
104     total_distance = 0
105     for i in range(len(path)-1):
106         node = path[i]
107         next_node = path[i+1]
108         edges = graph.edges[node]
109         for dest, distance in edges:
110             if dest == next_node:
111                 total_distance += distance
112                 break
113     return total_distance
114
115 def DFS(graph, start, end, path, shortest_path, shortest_distance, toPrint=False):
116     path = path + [start]
117     if toPrint and path[-1].getName() == "TIP":
118         print('Current DFS path:', printPath(path))
119     if start == end:

```