Hands-on Activity 1.3 | Transportation using Graphs

Objective(s):

This activity aims to demonstrate how to solve transportation related problem using Graphs

Intended Learning Outcomes (ILOs):

- Demonstrate how to compute the shortest path from source to destination using graphs
- Apply DFS and BFS to compute the shortest path

Resources:

- Jupyter Notebook

Procedures:

1. Create a Node class

```
class Node(object):
    def __init__(self, name):
        """Assumes name is a string"""
        self.name = name
    def getName(self):
        return self.name
    def __str__(self):
        return self.name
```

2. Create an Edge class

```
class Edge(object):
    def __init__(self, src, dest):
        """Assumes src and dest are nodes"""
        self.src = src
        self.dest = dest
    def getSource(self):
        return self.src
    def getDestination(self):
        return self.dest
    def __str__(self):
        return self.src.getName() + '->' + self.dest.getName()
```

3. Create Digraph class that add nodes and edges

```
class Digraph(object):
    """edges is a dict mapping each node to a list of
    its children"""
    def __init__(self):
        self.edges = {}
    def addNode(self, node):
        if node in self.edges:
            raise ValueError('Duplicate node')
        else:
            self.edges[node] = []
    def addEdge(self, edge):
        src = edge.getSource()
        dest = edge.getDestination()
        if not (src in self.edges and dest in self.edges):
            raise ValueError('Node not in graph')
        self.edges[src].append(dest)
    def childrenOf(self, node):
        return self.edges[node]
    def hasNode(self, node):
        return node in self.edges
    def getNode(self, name):
        for n in self.edges:
            if n.getName() == name:
                return n
        raise NameError(name)
    def __str__(self):
        result = ''
        for src in self.edges:
            for dest in self.edges[src]:
                result = result + src.getName() + '->'\
                         + dest.getName() + '\n'
        return result[:-1] #omit final newline
```

4. Create a Graph class from Digraph class that deifnes the destination and Source

```
class Graph(Digraph):
    def addEdge(self, edge):
        Digraph.addEdge(self, edge)
        rev = Edge(edge.getDestination(), edge.getSource())
        Digraph.addEdge(self, rev)
```

5. Create a buildCityGraph method to add nodes (City) and edges (source to destination)

```
def buildCityGraph(graphType):
    g = graphType()
    for name in ('Boston', 'Providence', 'New York', 'Chicago', 'Denver', 'Phoenix', 'Los Angeles'):
        #Create 7 nodes
        g.addNode(Node(name))
    g.addEdge(Edge(g.getNode('Boston'), g.getNode('Providence')))
    g.addEdge(Edge(g.getNode('Boston'), g.getNode('New York')))
    g.addEdge(Edge(g.getNode('Providence'), g.getNode('Boston')))
    g.addEdge(Edge(g.getNode('Providence'), g.getNode('New York')))
    g.addEdge(Edge(g.getNode('New York'), g.getNode('Chicago')))
    g.addEdge(Edge(g.getNode('Chicago'), g.getNode('Denver')))
    g.addEdge(Edge(g.getNode('Denver'), g.getNode('Phoenix')))
    g.addEdge(Edge(g.getNode('Denver'), g.getNode('New York')))
    g.addEdge(Edge(g.getNode('Los Angeles'), g.getNode('Boston')))
    return g


def printPath(path):
    """Assumes path is a list of nodes"""
    result = ''
    for i in range(len(path)):
        result = result + str(path[i])
        if i != len(path) - 1:
            result = result + '->'
    return result
```

6. Create a method to define DFS technique

```
def DFS(graph, start, end, path, shortest, toPrint = False):
    """Assumes graph is a Digraph; start and end are nodes;
          path and shortest are lists of nodes
       Returns a shortest path from start to end in graph"""
    path = path + [start]
    if toPrint:
        print('Current DFS path:', printPath(path))
    if start == end:
        return path
    for node in graph.childrenOf(start):
        if node not in path: #avoid cycles
            if shortest == None or len(path) < len(shortest):
                newPath = DFS(graph, node, end, path, shortest,
                              toPrint)
                if newPath != None:
                    shortest = newPath
        elif toPrint:
            print('Already visited', node)
    return shortest
```

7. Define a shortestPath method to return the shortest path from source to destination using DFS

```
def shortestPath(graph, start, end, toPrint = False):
    """Assumes graph is a Digraph; start and end are nodes
       Returns a shortest path from start to end in graph"""
    return DFS(graph, start, end, [], None, toPrint)
```

8. Create a method to test the shortest path method

```
def testSP(source, destination):
    g = buildCityGraph(Digraph)
    sp = shortestPath(g, g.getNode(source), g.getNode(destination),
                      toPrint = True)
    if sp != None:
        print('Shortest path from', source, 'to',
              destination, 'is', printPath(sp))
    else:
        print('There is no path from', source, 'to', destination)
```

9. Execute the testSP method

```
testSP('Boston', 'Phoenix')
```

```
Current DFS path: Boston
Current DFS path: Boston->Providence
Already visited Boston
Current DFS path: Boston->Providence->New York
Current DFS path: Boston->Providence->New York->Chicago
Current DFS path: Boston->Providence->New York->Chicago->Denver
Current DFS path: Boston->Providence->New York->Chicago->Denver->Phoenix
Already visited New York
Current DFS path: Boston->New York
Current DFS path: Boston->New York->Chicago
Current DFS path: Boston->New York->Chicago->Denver
Current DFS path: Boston->New York->Chicago->Denver->Phoenix
Already visited New York
Shortest path from Boston to Phoenix is Boston->New York->Chicago->Denver->Phoenix
```

Question:

Describe the DFS method to compute for the shortest path using the given sample codes

Base on my understanding is that we need to create ako graph that will visualize of both destination from it's current to goal destination which is stores the vertices and edges on an object.

10. Create a method to define BFS technique

```
def BFS(graph, start, end, toPrint = False):
    """Assumes graph is a Digraph; start and end are nodes
       Returns a shortest path from start to end in graph"""
    initPath = [start]
    pathQueue = [initPath]
    while len(pathQueue) != 0:
        #Get and remove oldest element in pathQueue
        tmpPath = pathQueue.pop(0)
        if toPrint:
            print('Current BFS path:', printPath(tmpPath))
        lastNode = tmpPath[-1]
        if lastNode == end:
            return tmpPath
        for nextNode in graph.childrenOf(lastNode):
            if nextNode not in tmpPath:
                newPath = tmpPath + [nextNode]
                pathQueue.append(newPath)
    return None
```

11. Define a shortestPath method to return the shortest path from source to destination using DFS

```
def shortestPath(graph, start, end, toPrint = False):
    """Assumes graph is a Digraph; start and end are nodes
       Returns a shortest path from start to end in graph"""
    return BFS(graph, start, end, toPrint)
```

12. Execute the testSP method

```
testSP('Boston', 'Phoenix')
```

```
Current BFS path: Boston
Current BFS path: Boston->Providence
Current BFS path: Boston->New York
Current BFS path: Boston->Providence->New York
Current BFS path: Boston->New York->Chicago
Current BFS path: Boston->Providence->New York->Chicago
Current BFS path: Boston->New York->Chicago->Denver
Current BFS path: Boston->Providence->New York->Chicago->Denver
Current BFS path: Boston->New York->Chicago->Denver->Phoenix
Shortest path from Boston to Phoenix is Boston->New York->Chicago->Denver->Phoenix
```

Question:

Describe the BFS method to compute for the shortest path using the given sample codestion: Describe the BFS method to compute for the shortest path using the given sample codestion:

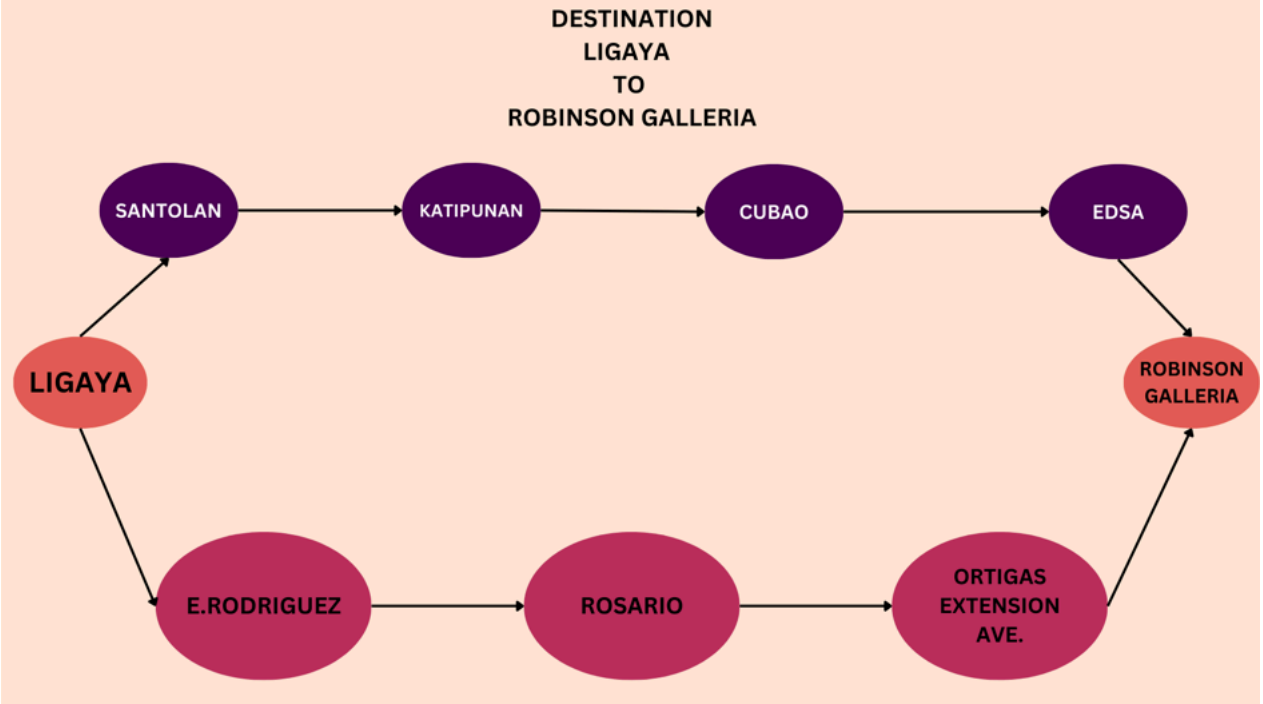
This BFS is also same use of bottom up approach algorithm.

Supplementary Activitiy

- Use a specific location or city to solve transportation using graph

  Marikina City, Ligaya to Robinson Galleria in Pasig City

- Use DFS and BFS methods to compute the shortest path
- Display the shortest path from source to destination using DFS and BFS
- Differentiate the performance of DFS from BFS


DESTINATION
LIGAYA
TO
ROBINSON GALLERIA
SANTOLAN
KATIPUNAN
CUBAO
EDSA
LIGAYA
ROBINSON GALLERIA
E.RODRIGUEZ
ROSARIO
ORTIGAS EXTENSION AVE.

```
class Node(object):
    def __init__(self, name):

        self.name = name
    def getName(self):
        return self.name
    def __str__(self):
        return self.name
class Edge(object):
    def __init__(self, src, dest):

        self.src = src
        self.dest = dest
    def getSource(self):
        return self.src
    def getDestination(self):
        return self.dest
    def __str__(self):
        return self.src.getName() + '->' + self.dest.getName()

class Digraph(object):

    def __init__(self):
        self.edges = {}
    def addNode(self, node):
        if node in self.edges:
            raise ValueError('Duplicate node')
        else:
            self.edges[node] = []
    def addEdge(self, edge):
        src = edge.getSource()
        dest = edge.getDestination()
        if not (src in self.edges and dest in self.edges):
            raise ValueError('Node not in graph')
        self.edges[src].append(dest)
    def childrenOf(self, node):
        return self.edges[node]
    def hasNode(self, node):
        return node in self.edges
    def getNode(self, name):
        for n in self.edges:
            if n.getName() == name:
                return n
        raise NameError(name)
    def __str__(self):
        result = ''
        for src in self.edges:
            for dest in self.edges[src]:
                result = result + src.getName() + '->'\
                         + dest.getName() + '\n'
        return result[:-1]
class Graph(Digraph):
    def addEdge(self, edge):
        Digraph.addEdge(self, edge)
        rev = Edge(edge.getDestination(), edge.getSource())
        Digraph.addEdge(self, rev)
def buildCityGraph(graphType):
    g = graphType()
    for name in ('Ligaya','Santolan','Katipunan','Cubao','Edsa','Robinson Galleria','E.rodriguez','Rosario','Ortigas Ext ave.'):
        #Create 7 nodes
        g.addNode(Node(name))
    g.addEdge(Edge(g.getNode('Ligaya'), g.getNode('E.rodriguez')))
    g.addEdge(Edge(g.getNode('E.rodriguez'), g.getNode('Rosario')))
    g.addEdge(Edge(g.getNode('Rosario'), g.getNode('Ortigas Ext ave.')))
    g.addEdge(Edge(g.getNode('Ortigas Ext ave.'), g.getNode('Robinson Galleria')))
    g.addEdge(Edge(g.getNode('Ligaya'), g.getNode('Santolan')))
    g.addEdge(Edge(g.getNode('Santolan'), g.getNode('Katipunan')))
    g.addEdge(Edge(g.getNode('Katipunan'), g.getNode('Cubao')))
    g.addEdge(Edge(g.getNode('Cubao'), g.getNode('Edsa')))
    g.addEdge(Edge(g.getNode('Edsa'), g.getNode('Robinson Galleria')))

    return g

def printPath(path):

    result = ''
    for i in range(len(path)):
```

```
            result = result + str(path[i])
            if i != len(path) - 1:
                result = result + '->'

    def BFS(graph, start, end, toPrint = False):

        initPath = [start]
        pathQueue = [initPath]
        while len(pathQueue) != 0:

            tmpPath = pathQueue.pop(0)
            if toPrint:
                print('Your Current Destination is:', printPath(tmpPath))
            lastNode = tmpPath[-1]
            if lastNode == end:
                return tmpPath
            for nextNode in graph.childrenOf(lastNode):
                if nextNode not in tmpPath:
                    newPath = tmpPath + [nextNode]
                    pathQueue.append(newPath)
        return None

    def shortestPath(graph, start, end, toPrint = False):

        return BFS(graph, start, end, toPrint)


    testSP('Ligaya', 'Robinson Galleria')
```

```
Your Current Destination is: Ligaya
Your Current Destination is: Ligaya->E.rodriguez
Your Current Destination is: Ligaya->Santolan
Your Current Destination is: Ligaya->E.rodriguez->Rosario
Your Current Destination is: Ligaya->Santolan->Katipunan
Your Current Destination is: Ligaya->E.rodriguez->Rosario->Ortigas Ext ave.
Your Current Destination is: Ligaya->Santolan->Katipunan->Cubao
Your Current Destination is: Ligaya->E.rodriguez->Rosario->Ortigas Ext ave.->Robinson Galleria
The Shortest way to travel from Ligaya to Robinson Galleria is Ligaya->E.rodriguez->Rosario->Ortigas Ext ave.->Robinson Galleria
```

```
testSP('Ligaya', 'Robinson Galleria')
```

Type your evaluation about the performance of DFS and BFS

```
Your Current Destination is: Ligaya->E.rodriguez->Rosario
```

Conclusion

For my Evaluation for the both perfomance is that they both running well and fast since the code are already been provided for this hands on activity

for the DFS it does a lot of time to take a process since it's very complex code while the BFS does not take a lot time since the it doesn't have a lot of code and it's uses while looping implemantion for this.

. .