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
        #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')))
    g.addEdge(Edge(g.getNode('Phoenix'), g.getNode('Alabama')))
    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:

Answer:

The Depth-First Search method (DFS) for computing the shortest path utilizes recursion in finding the shortest path. By creating a variable that stores the current shortest path found by the recursion algorithm, we can iterate and search through the entire graph, stopping at the specified endpoint and then compare all possible paths to reach it and get the shortest one. By saving the current shortest path outside of the recursion, we will be able to determine the shortest path despite finding a new path from the recursion. DFS Method always tries to check any existing children nodes until it reaches the end node, which means we can immediately find the shortest path depending on how linear and directed the graph is. However, despite us being able to find the shortest path easily, the algorithm will still iterate through the entire graph. While this seems inefficient, this makes sure that all possible paths have been computed and we do get the shortest path possible. By saving the current shortest path throughout the recursion, we will be able to check and perform the DFS without having any problems with any order of checking or any exceeding nodes past the end node.

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 codes:

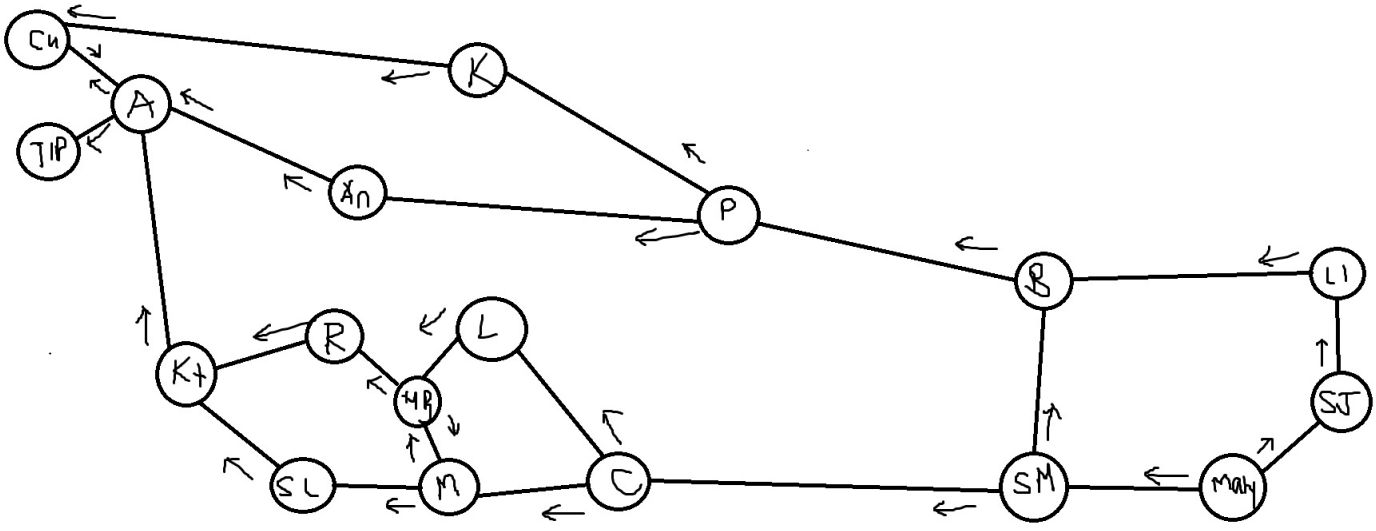
Answer:

The Breadth-First Search method (BFS) for computing the shortest path utilizes a queue data structure to store every adjacent node to the current node. This means that it treats the starting node as a center, spanning outwards in levels when searching. The nearer the end node is to the start node, the faster the BFS computation for shortest path is. We can stop the computation as soon as we arrive at the end node since we are computing from the shortest possible starting path. As soon as we find the end node, we can be sure that it is the shortest path possible. However, the only downside to it is that when the final node is very far from the start, it will take as much time as with the DFS method.

Supplementary Activitiy

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

Jem lives in Maly, San Mateo. He is studying at TIP QC. From where he lives, There is a number of ways that he can travel in to get to his campus. We can use a directed graph to represent all the landmarks and paths that determine the routes that he can take.



```
# new location graph
# creates a graph of all possible routes from Maly to TIP

def buildGraph(graphType):
    g = graphType()
    for name in ('Maly', 'San Jose', 'LITEX', 'Batasan', 'Philcoa', #create 18 nodes
                 'Kalayaan Ave.', 'Araneta-Cubao', 'Aurora Blvd.',
                 'TIP', 'Anonas', 'SM San Mateo', 'Concepcion',
                 'Lamuan', 'Marikina-Bayan', 'Marquinton', 'Sta. Lucia',
                 'Katipunan Ave.', 'Riverbanks'):
        g.addNode(Node(name))

    # generate directed edges
    g.addEdge(Edge(g.getNode('Maly'), g.getNode('San Jose')))
    g.addEdge(Edge(g.getNode('San Jose'), g.getNode('LITEX')))
    g.addEdge(Edge(g.getNode('LITEX'), g.getNode('Batasan')))
    g.addEdge(Edge(g.getNode('Batasan'), g.getNode('Philcoa')))
    g.addEdge(Edge(g.getNode('Philcoa'), g.getNode('Kalayaan Ave.')))
    g.addEdge(Edge(g.getNode('Kalayaan Ave.'), g.getNode('Araneta-Cubao')))
    g.addEdge(Edge(g.getNode('Araneta-Cubao'), g.getNode('Aurora Blvd.')))
    g.addEdge(Edge(g.getNode('Aurora Blvd.'), g.getNode('TIP')))
    g.addEdge(Edge(g.getNode('Philcoa'), g.getNode('Anonas')))
    g.addEdge(Edge(g.getNode('Anonas'), g.getNode('Aurora Blvd.')))
    g.addEdge(Edge(g.getNode('Maly'), g.getNode('SM San Mateo')))
    g.addEdge(Edge(g.getNode('SM San Mateo'), g.getNode('Batasan')))
    g.addEdge(Edge(g.getNode('SM San Mateo'), g.getNode('Concepcion')))
    g.addEdge(Edge(g.getNode('Concepcion'), g.getNode('Lamuan')))
    g.addEdge(Edge(g.getNode('Concepcion'), g.getNode('Marquinton')))
    g.addEdge(Edge(g.getNode('Lamuan'), g.getNode('Marikina-Bayan')))
    g.addEdge(Edge(g.getNode('Marikina-Bayan'), g.getNode('Marquinton')))
    g.addEdge(Edge(g.getNode('Marquinton'), g.getNode('Marikina-Bayan')))
    g.addEdge(Edge(g.getNode('Marikina-Bayan'), g.getNode('Riverbanks')))
    g.addEdge(Edge(g.getNode('Riverbanks'), g.getNode('Katipunan Ave.')))
    g.addEdge(Edge(g.getNode('Marquinton'), g.getNode('Sta. Lucia')))
    g.addEdge(Edge(g.getNode('Sta. Lucia'), g.getNode('Katipunan Ave.')))
    g.addEdge(Edge(g.getNode('Katipunan Ave.'), g.getNode('Aurora Blvd.')))

    return g


```

```
# type your code here using DFS

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

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)

def testSP(source, destination):
    g = buildGraph(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)

testSP('Maly', 'TIP')
```

```
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->Kalayaan Ave.
Current DFS path: Maly->San Jose->LITEX->Batasan->Philcoa->Kalayaan Ave.->Araneta-Cubao
Current DFS path: Maly->San Jose->LITEX->Batasan->Philcoa->Kalayaan Ave.->Araneta-Cubao-
Current DFS path: Maly->San Jose->LITEX->Batasan->Philcoa->Kalayaan Ave.->Araneta-Cubao-
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->Kalayaan Ave.
Current DFS path: Maly->SM San Mateo->Batasan->Philcoa->Kalayaan Ave.->Araneta-Cubao
Current DFS path: Maly->SM San Mateo->Batasan->Philcoa->Kalayaan Ave.->Araneta-Cubao->Au
Current DFS path: Maly->SM San Mateo->Batasan->Philcoa->Kalayaan Ave.->Araneta-Cubao->Au
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->Marquinton
Already visited Marikina-Bayan
Current DFS path: Maly->SM San Mateo->Concepcion->Lamuan->Marikina-Bayan->Marquinton->St
Current DFS path: Maly->SM San Mateo->Concepcion->Lamuan->Marikina-Bayan->Riverbanks
Current DFS path: Maly->SM San Mateo->Concepcion->Lamuan->Marikina-Bayan->Riverbanks->Ka
Current DFS path: Maly->SM San Mateo->Concepcion->Marquinton
Current DFS path: Maly->SM San Mateo->Concepcion->Marquinton->Marikina-Bayan
Already visited Marquinton
Current DFS path: Maly->SM San Mateo->Concepcion->Marquinton->Marikina-Bayan->Riverbanks
Current DFS path: Maly->SM San Mateo->Concepcion->Marquinton->Marikina-Bayan->Riverbanks
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.
Shortest path from Maly to TIP is Maly->SM San Mateo->Batasan->Philcoa->Anonas->Aurora B
```

```
# type your code here using BFS

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)

testSP('Maly', 'TIP')
```

```
Current BFS path: Maly
Current BFS path: Maly->San Jose
Current BFS path: Maly->SM San Mateo
Current BFS path: Maly->San Jose->LITEX
Current BFS path: Maly->SM San Mateo->Batasan
Current BFS path: Maly->SM San Mateo->Concepcion
Current BFS path: Maly->San Jose->LITEX->Batasan
Current BFS path: Maly->SM San Mateo->Batasan->Philcoa
Current BFS path: Maly->SM San Mateo->Concepcion->Lamuan
Current BFS path: Maly->SM San Mateo->Concepcion->Marquinton
Current BFS path: Maly->San Jose->LITEX->Batasan->Philcoa
Current BFS path: Maly->SM San Mateo->Batasan->Philcoa->Kalayaan Ave.
Current BFS path: Maly->SM San Mateo->Batasan->Philcoa->Anonas
Current BFS path: Maly->SM San Mateo->Concepcion->Lamuan->Marikina-Bayan
Current BFS path: Maly->SM San Mateo->Concepcion->Marquinton->Marikina-Bayan
Current BFS path: Maly->SM San Mateo->Concepcion->Marquinton->Sta. Lucia
Current BFS path: Maly->San Jose->LITEX->Batasan->Philcoa->Kalayaan Ave.
```

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

Type your evaluation about the performance of DFS and BFS

From the results of the 2 Search Methods for finding the Shortest Path, We can see that BFS was able to find the shortest path faster than the DFS. This is because when BFS is used, we will try all possible paths from the start point at a level basis. This means that we take one step from each possible path every time. Once we reach the end point for the first time, we can already say that it is the shortest path since all paths that we are traversing are moving along at the same rate as each other. With DFS however, it won't be able to tell that it has the shortest path. The entire search has to finish before we can be sure that it has found the shortest path. In terms of time and efficiency, BFS