COMP2119 Assignment 3 Code Report

Brief idea:

3 functions are created.

- Creating_edges(): to create edges between the nodes
- Build_graph(): to make a graph, which contains each nodes and connected nodes using dictionary
- leastNumBus(): using queue, analyze paths

Using case 1 as example:

- Creating_edges(routes):

  [[5, 0], [6, 0], [7, 1], [7, 0], [1, 0], [8, 1], [8, 0], [1, 0], [9, 1]]
- Build_graph(routes):

  {

  0: [5,6,7,1,8],

  1: [7,0,8,9],

  5: [0],

  6: [0],

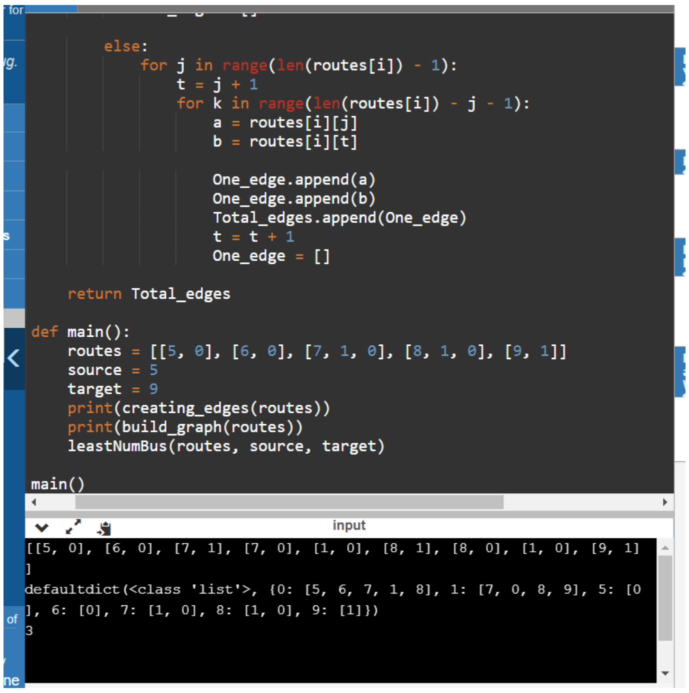
  7: [1,0],

  8: [1,0],

  9: [1]

  }
- leastNumBus(routes, source, target)

  3



```python
        else:
            for j in range(len(routes[i]) - 1):
                t = j + 1
                for k in range(len(routes[i]) - j - 1):
                    a = routes[i][j]
                    b = routes[i][t]

                    One_edge.append(a)
                    One_edge.append(b)
                    Total_edges.append(One_edge)
                    t = t + 1
                    One_edge = []

    return Total_edges

def main():
    routes = [[5, 0], [6, 0], [7, 1, 0], [8, 1, 0], [9, 1]]
    source = 5
    target = 9
    print(creating_edges(routes))
    print(build_graph(routes))
    leastNumBus(routes, source, target)

main()
```

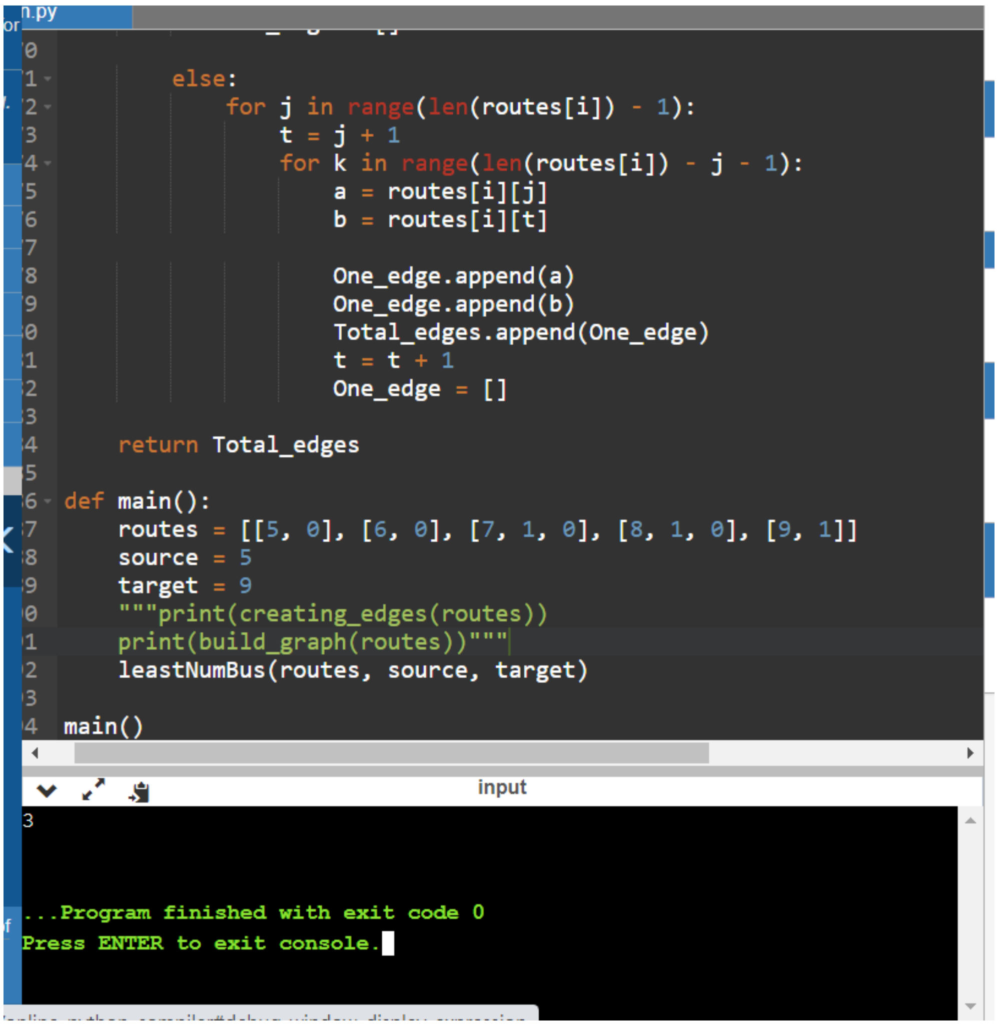
```
[[5, 0], [6, 0], [7, 1], [7, 0], [1, 0], [8, 1], [8, 0], [1, 0], [9, 1]
]
defaultdict(<class 'list'>, {0: [5, 6, 7, 1, 8], 1: [7, 0, 8, 9], 5: [0
], 6: [0], 7: [1, 0], 8: [1, 0], 9: [1]})
3
```

The following is how I test the code (using case 1 as example)

```
def leastNumBus( routes, source, target):
    """
    :type routes: List[List[int]]
    :type source: int
    :type target: int
    :rtype: int
    """
    graph = build_graph(routes)
    explored = []
    queue = [[source]]

    while queue:
        path = queue.pop(0)
        node = path[-1]

        if node not in explored:
            neighbours = graph[node]

            for neighbour in neighbours:
                new_path = list(path)
                new_path.append(neighbour)
                queue.append(new_path)

                if neighbour == target:
                    print(len(new_path)-1)
                    return
            explored.append(node)

    print("-1")
    return

from collections import defaultdict

def build_graph(routes):
    edges_not_sorted = creating_edges(routes)
    edges = []
    for element in edges_not_sorted:
        if element not in edges:
            edges.append(element)

    graph = defaultdict(list)

    for edge in edges:
        a, b = edge[0], edge[1]

        graph[a].append(b)
        graph[b].append(a)

    return graph

def creating_edges(routes):

    i = 0
    j = 0
    k = 0
    One_edge = []
    Total_edges = []

    for i in range(len(routes)):
        if len(routes[i]) == 2:
            a = routes[i][0]
            b = routes[i][1]

            One_edge.append(a)
            One_edge.append(b)

            Total_edges.append(One_edge)
            One_edge = []

        else:
            for j in range(len(routes[i]) - 1):
                t = j + 1
                for k in range(len(routes[i]) - j - 1):
                    a = routes[i][j]
                    b = routes[i][t]

                    One_edge.append(a)
                    One_edge.append(b)
                    Total_edges.append(One_edge)
                    t = t + 1
                    One_edge = []

    return Total_edges

def main():
    routes = [[5, 0], [6, 0], [7, 1, 0], [8, 1, 0], [9, 1]]
    source = 5
    target = 9
    “””print(creating_edges(routes))
    print(build_graph(routes))”””
    leastNumBus(routes, source, target)

main()
```

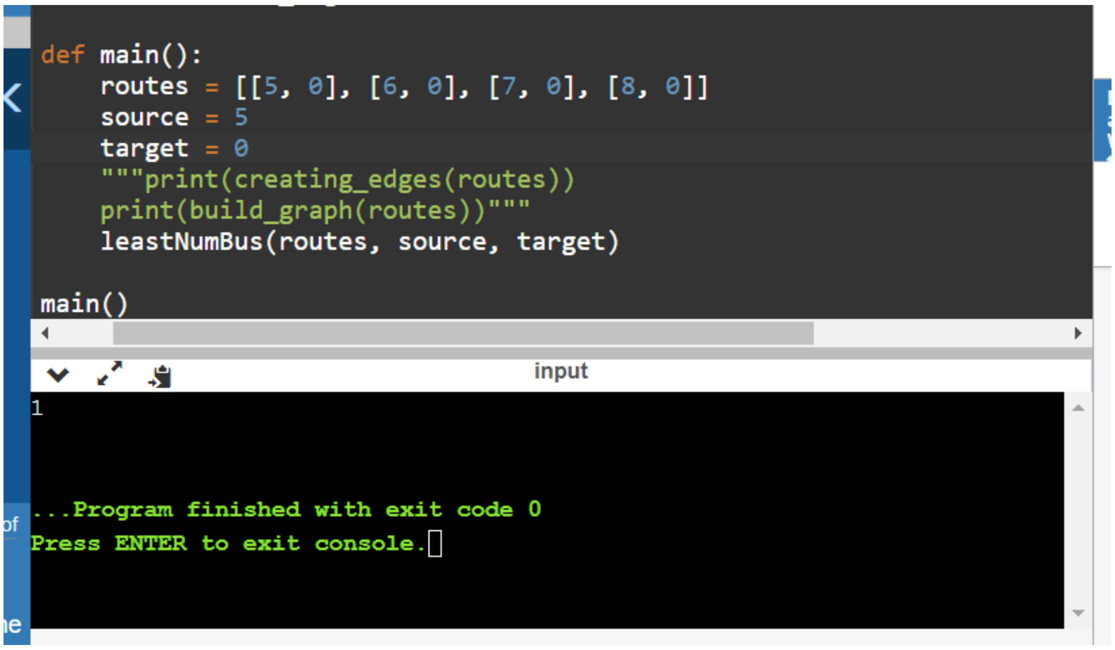
Results:

The following are screenshots of case 1 – 5.



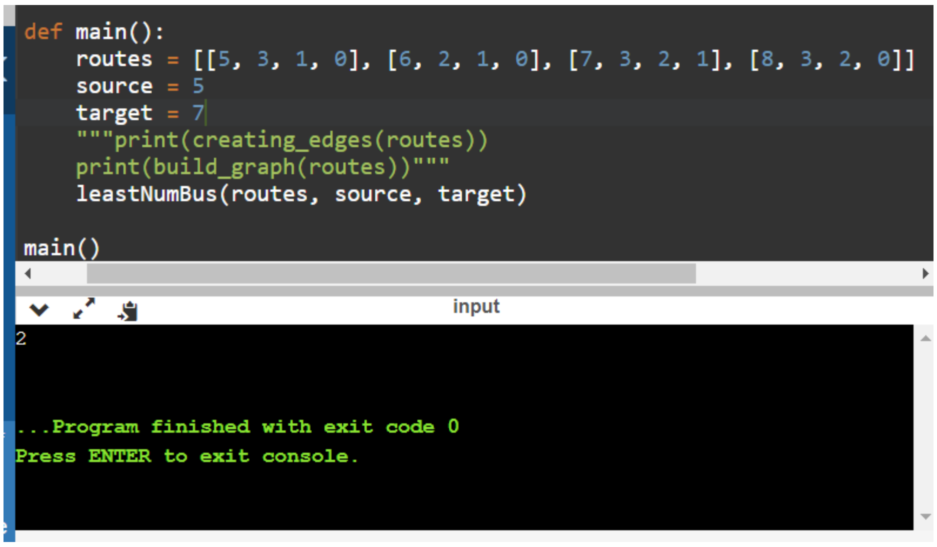
```
        else:
            for j in range(len(routes[i]) - 1):
                t = j + 1
                for k in range(len(routes[i]) - j - 1):
                    a = routes[i][j]
                    b = routes[i][t]

                    One_edge.append(a)
                    One_edge.append(b)
                    Total_edges.append(One_edge)
                    t = t + 1
                    One_edge = []

    return Total_edges

def main():
    routes = [[5, 0], [6, 0], [7, 1, 0], [8, 1, 0], [9, 1]]
    source = 5
    target = 9
    """print(creating_edges(routes))
    print(build_graph(routes))"""
    leastNumBus(routes, source, target)

main()
```

```
3


...Program finished with exit code 0
Press ENTER to exit console.
```



```
def main():
    routes = [[5, 0], [6, 0], [7, 0], [8, 0]]
    source = 5
    target = 0
    """print(creating_edges(routes))
    print(build_graph(routes))"""
    leastNumBus(routes, source, target)

main()
```

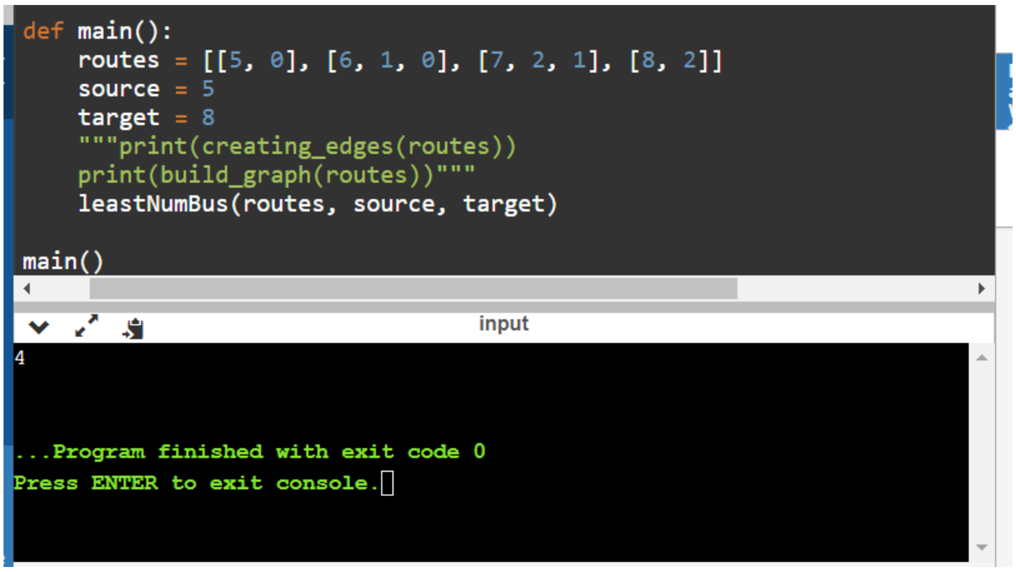
```
1


...Program finished with exit code 0
Press ENTER to exit console.
```



```
def main():
    routes = [[5, 3, 1, 0], [6, 2, 1, 0], [7, 3, 2, 1], [8, 3, 2, 0]]
    source = 5
    target = 7
    """print(creating_edges(routes))
    print(build_graph(routes))"""
    leastNumBus(routes, source, target)

main()
```

input

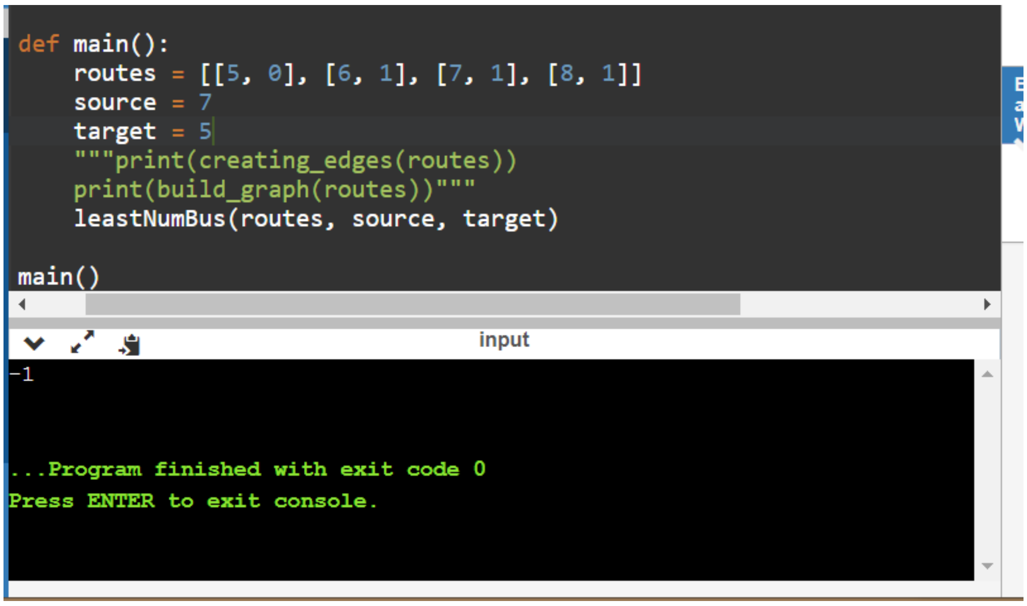
```
2


...Program finished with exit code 0
Press ENTER to exit console.
```



```
def main():
    routes = [[5, 0], [6, 1, 0], [7, 2, 1], [8, 2]]
    source = 5
    target = 8
    """print(creating_edges(routes))
    print(build_graph(routes))"""
    leastNumBus(routes, source, target)

main()
```

input

```
4


...Program finished with exit code 0
Press ENTER to exit console.
```



```
def main():
    routes = [[5, 0], [6, 1], [7, 1], [8, 1]]
    source = 7
    target = 5
    """print(creating_edges(routes))
    print(build_graph(routes))"""
    leastNumBus(routes, source, target)

main()
```

input

```
-1


...Program finished with exit code 0
Press ENTER to exit console.
```