ICN HW2

B09901066 謝承修

註：我在程式裡面都有打註解，如果直接看程式碼應該也沒問題。

Part 1

First, we have to read file, to store the input data with a list "data" and record the number of nodes in "node".

```
node = 0  # num of nodes
data = []  # input data
########## read file ##########
with open(inputfile, 'r') as f:
    line = f.readline()
    node = int(line)
    while line:
        line = f.readline()
        eachline = line.split()
        num = [int(x) for x in eachline[0:]]
        n = np.array(num)
        data.append(n)
    if (removed_router_number != -1):
        for i in range(node):
            data[removed_router_number][i] = -1
            data[i][removed_router_number] = -1
```

Then, we can open a file to write down the output and start to run the Dijkstra's Algorithm.

```
########## start to run ##########
output = open(outputfile, "w")
for i in range(node):
    e = copy.deepcopy(data)
    if i != removed_router_number:
        dijkstra(e, i, output)
output.close()
```

In the algorithm, I first initialize the "dist" and "nextHop".

```
    # initialize the distance and nextHop
    dist = e[source]
    pred = np.ones(node, dtype=int)*(-1)
    nextHop = np.ones(node, dtype=int)*(-1)
    nextHop[source] = source
```

And I create two lists, leftNode(nodes which are not added to final graph yet) and queue(nodes which are reached but not added to final graph yet). After adding the source node to final graph, we start the while loop, to add nodes in queue to final graph one by one(start from the minimum distance), and update the distance for all left nodes. As a node is added, we remove it from leftNode and queue.

```
# early stop for the isolated node
    if np.count_nonzero(dist == -1) != node-1:

        # leftNode: nodes which are not added to final graph yet
        # queue: nodes which are reached but not added to final graph yet
        leftNode = [int(x) for x in range(node)]
        leftNode.remove(source)
        if removed_router_number != -1:
            leftNode.remove(removed_router_number)

        queue = set()
        for i in leftNode:
            if dist[i] != -1:
                queue.add(i)
                nextHop[i] = i
        while queue:

            # find the node with minimum distance from source
            min = np.inf
            u = -1
            for i in queue:
                if dist[i] != -1 and dist[i] < min:
                    min = dist[i]
                    u = i
            # update distance of all left nodes
            if u != -1:
                queue.remove(u)  # u is added to final graph
                leftNode.remove(u)
                for v in leftNode:
                    if e[u][v] != -1 and (dist[v] == -1 or dist[v] > dist[u]+e[u][v]):
                        dist[v] = dist[u]+e[u][v]
                        nextHop[v] = nextHop[u]
                        queue.add(v)

    printResult(source, dist, nextHop, output)
```

After run the algorithm, we get the “dist” and “nextHop”, and thus we can output the result.

```
def printResult(source, cost, next, output):

    # reindex the node number
    output.write("Routing table of router {}:".format(source+1))
    output.write('\n')
    for i in range(node):
        output.write(str(cost[i]))
        output.write(' ')
        if next[i] == -1:
            output.write(str(next[i]))
        else:
            output.write(str(next[i]+1))
        output.write('\n')
```

Part 2

We have to process the command arguments. To differentiate between part1 and part2, we first consider the argv length, if sys.argv[2] exists, we let “removed_router_number” be meaningful.

```
if len(sys.argv) <= 1:
    print('Usage : "python hw2.py p2_test1.txt [Removed Router Number]"')
    sys.exit(2)

inputfile = sys.argv[1]
removed_router_number = -1 if len(sys.argv) < 3 else int(sys.argv[2])-1
if (removed_router_number == -1):
    outputfile = inputfile.replace(".txt", "_GenTable.txt")
else:
    outputfile = inputfile.replace(
        ".txt", "_RmRouter"+sys.argv[2]+".txt")
```

And in the read file procedure, we have to change the distance between all nodes and the removed node to -1, to represent they are not connected.

```
    if (removed_router_number != -1):
        for i in range(node):
            data[removed_router_number][i] = -1
            data[i][removed_router_number] = -1
```

We run Dijkstra's algorithm only when the source are not the removed one.

```
if i != removed_router_number:
        dijkstra(e, i, output)
```

In the algorithm, if there's a removed node, we consider the case of early stop.

```
    # early stop for the isolated node
    if np.count_nonzero(dist == -1) != node-1:
```

And also remove the removed node from "leftNode".

```
if removed_router_number != -1:
            leftNode.remove(removed_router_number)
```

The remain part is the same as previous case, the loop always terminate when "queue" is empty.

註：我有在最後面多打了一段程式用來檢查產生出的結果是否和 golden 相同，六個答案都是一樣的。

```
# compare the result with the golden file
if (removed_router_number == -1):
    print(filecmp.cmp(outputfile, inputfile.replace(".txt", "_golden.txt")))
else:
    print(filecmp.cmp(outputfile, inputfile.replace(
        ".txt", "_golden_rm"+sys.argv[2]+".txt").replace("p1", "p2")))
```