學號：0653422　　科目：計算機網路 HW1　　姓名：林容伊

使用的程式語言：python

inputs:

- Network topology (由使用者輸入的參數形成 graph)

  - 請輸入全部共有幾個 nodes（routers）

```
In [376]: runfile('/Users/apple/Documents/01.-
Decision-Tree/net_HW.py', wdir='/Users/apple/
Documents/01.-Decision-Tree')

router numbers(please input 1 ~ 5):
```

請輸入 5

  - 再輸入每個 nodes 有幾個鄰居

  - node 1 到 node 5 的鄰居是誰？ weight 是多少？（ weight=link cost ）

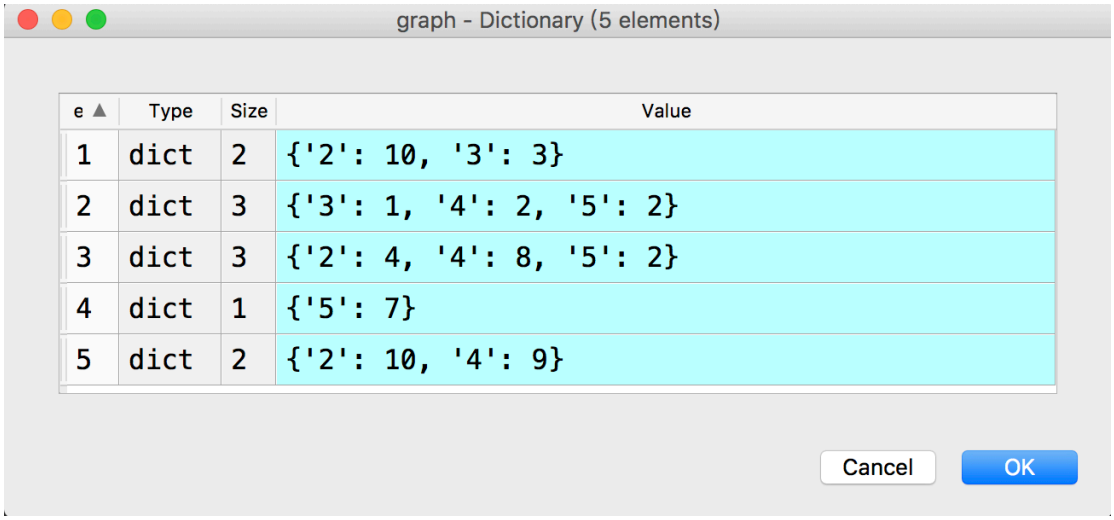
(例如：輸入 node 1 有兩個 neighbor，那分別輸入 node 1 的第一個鄰居是 2，然後他的 weight 是 10、第二個鄰居是 3，他的 weight 是 3)

```
router numbers(please input 1 ~ 5): 5

how many neighbors does the node 1 have: 2

Please enter the neighbor of node 1 : 2

Please enter the link cost: 10

Please enter the neighbor of node 1 : 3

Please enter the link cost: 3

how many neighbors does the node 2 have:
```

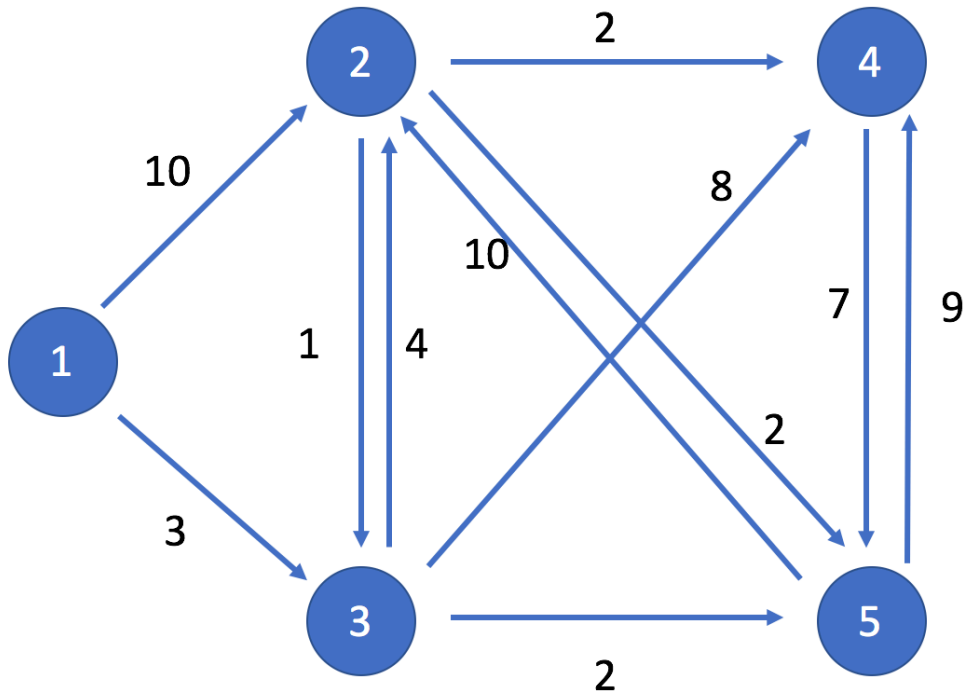
  - user 全部輸入完 graph 就畫出來了

p.s graph 會以 dictionary 方式呈現：

graph = {'1':{'2':10,'3':3},'2':{'3':1,'4':2,'5':2},'3':{'2':4,'4':8,'5':2},'4':{'5':7},'5':{'2':10,'4':9}}



-如下圖所示：



- Router Requests
    - 請輸入一開始要走的 node 跟結束的 node 還有 capacity demand（請直接一行就輸入所有參數 用空白鍵隔開~~）

```
In [377]: runfile('/Users/apple/Documents/01.-
Decision-Tree/net_HW.py', wdir='/Users/apple/
Documents/01.-Decision-Tree')

Router Request:1 5 1
```

(上圖由左至右為 source router：1　destination router：5　capacity demand：1)

- user 可以一直輸入 Router Request 直到以下情形發生的時候：
    - 注意：如果 Router Request 的第一個參數 輸入 0 X X(X 表示其他非 0 的參數) 則就會跳出迴圈

```
Router Request:0 5 1
no path to route
```

outputs:

- Discovered path of each route request
    - Shortest Path 會印出一條最短路徑（程式以 dijkestra 跑）
    - Shortest Distance 會印出最短的路徑長度
    - All paths 會印出所有可以走的路徑

-Satisfaction index：印出最短的一條 / 所有可能路徑

```
Router Request:1 5 2
Shortest Distance: 5
Shortest Path: ['1', '3', '5']
All paths:  [['1', '2', '3', '4', '5'], ['1', '2', '3', '5'],
['1', '2', '4', '5'], ['1', '2', '5'], ['1', '3', '2', '4',
'5'], ['1', '3', '2', '5'], ['1', '3', '4', '5'], ['1', '3',
'5']]
Satisfaction index:  0.125

Router Request:1 4 3
Shortest Distance: 11
Shortest Path: ['1', '3', '4']
All paths:  [['1', '2', '3', '4'], ['1', '2', '3', '5', '4'],
['1', '2', '4'], ['1', '2', '5', '4'], ['1', '3', '2', '4'],
['1', '3', '2', '5', '4'], ['1', '3', '4'], ['1', '3', '5',
'2', '4'], ['1', '3', '5', '4']]
Satisfaction index:  0.1111111111111111
```

- 如果 user 輸入的 demend，graph 裡面的 weight(link capacity)小於 demand，那條路就不能走了

- 如果起點 start 走不到 end，則會顯示 No path to route

```
Router Request:1 4 4
no path to route

Router Request:
```

程式說明：

1. Def dijkestra：(先設定好變數)

```
def dijkestra(graph,start,end,demend):
    graph2 = graph.copy()
    shortest_dist = {} #開一個空的dictionary存最短距離
    parent = {} #記錄各個點在最短路徑上的父親是誰
    unseenNodes = graph2 #還沒拜訪過的node們
    inf = 999999 #將還沒拜訪過的distance先設無限大
    path = [] #存放未來最短路徑的path
```

2. 設定 minnode：(第一個要開始跑的 node)

```
    for i in unseenNodes:
        shortest_dist[i] = inf #還沒拜訪過的node的shortest_dist會先設成無限大
    shortest_dist[start] = 0 #將start的最短路徑先設成0

    while unseenNodes: #loop一直到unseenNodes裡面每個node都拜訪過才跳出迴圈
        minNode = None
        for i in unseenNodes: #要挑出有最小的distance來當我第一個要走的node
            if minNode == None: #best case
                minNode = i
            elif shortest_dist[i] < shortest_dist[minNode]: #如果有出現distance
                minNode = i
```

3. 用 minNode 來開始跑 dijkestra 來更新 shortest_dist
   - 當我的 link capacity 小於 capacity demand 表示那條路出現 bottleneck 那條路便不能走了，所以 del parent[neighbor] 是把當前 node 的 neighbor 從 parent dictionary 紀錄裡面刪掉

```
31
32      #選好minNode之後更新shortest_dist
33       for neighbor,weight in graph[minNode].items():
34
35           if weight + shortest_dist[minNode] < shortest_dist[neighbor] and weight >= demend:
36               shortest_dist[neighbor] = weight + shortest_dist[minNode]
37               parent[neighbor] = minNode
38               if weight < demend: #如果weight(link capacity)小於 demend 這條就不能走
39                   del parent[neighbor] #就把這個neighbor node 紀錄刪掉
40                   parent[neighbor] = minNode
41                   shortest_dist[neighbor] = shortest_dist[neighbor] - weight
42       #print(parent) #可以知道目前所有node的parent是誰
43
44       unseenNodes.pop(minNode) #把已經跑完的minNode從unseenNode裡面pop掉
45
```

4. 再來要把最短路徑 insert 進去 path 裡面(每次加進去都會放在前面) 從 end(destination node 往前推)
   - current_check 用來確認我現在的 parent node 的 key 裡面存不存在終點 node(終點 node 一定要存在不然最短路線變不存在了)
   - start_check 用來確認我現在的 parent node 的 value 有沒有起點 node(如果是 False 表示沒有 start node 可以走到終點 node)
   - 兩者只要有一個不成立便印出 no path to route

```
45
46  current = end
47  while current != start: #從current倒推回起點
48      current_check = current in parent.keys() #檢查current node有沒有在parent key裡面
49      start_check = start in parent.values() #檢查start node有沒有在parent value裡面
50      if current_check == False or start_check == False: #如果其中有一個不符合表示沒有路可以走了
51          print('no path to route')
52          break
53
```

5. Def Findallpath: 會找到 graph 所有 start node 走到 destination node 的所有路徑

```
67
68 def Findallpath(graph,start,end,path=[]): #找到start到destination的所有路徑
69     path = path + [start] #把起點加到path裡面
70     if start == end:
71         return [path]
72     if start not in graph: #如果start不再graph裡面
73             return [] #回傳空的list
74     paths = []
75     for i in graph[start]: #graph dictionary裡面的第一個value{'2':10,'3':3}
76         if i not in path: #i是graph第一排裡面的key
77             newpath = Findallpath(graph,i,end,path) #會不斷的在graph裡面往下找
78             for j in newpath:
79                 paths.append(j)
80     return paths
81
```

6. 最後的 output 最佳情況會找到一條最短路徑，最差情況是走不到

```
Router Request:1 4 1
Shortest Distance: 9
Shortest Path: ['1', '3', '2', '4']
All paths:  [['1', '2', '3', '4'], ['1', '2', '3', '5', '4'], ['1',
'2', '4'], ['1', '2', '5', '4'], ['1', '3', '2', '4'], ['1', '3', '2',
'5', '4'], ['1', '3', '4'], ['1', '3', '5', '2', '4'], ['1', '3', '5',
'4']]
Satisfaction index:  0.1111111111111111

Router Request:1 4 3
Shortest Distance: 11
Shortest Path: ['1', '3', '4']
All paths:  [['1', '2', '3', '4'], ['1', '2', '3', '5', '4'], ['1',
'2', '4'], ['1', '2', '5', '4'], ['1', '3', '2', '4'], ['1', '3', '2',
'5', '4'], ['1', '3', '4'], ['1', '3', '5', '2', '4'], ['1', '3', '5',
'4']]
Satisfaction index:  0.1111111111111111

Router Request:1 4 4
no path to route

Router Request:
```