期末上機程式碼說明 大氣4A 106601015 黃展皇

這次的期末上機報告中我選擇Dijkstra演算法作為報告講解內容，完整程式碼如Dijkstra.py所示。

Dijkstra's algorithm

Input

首先第一行會輸入兩個字元(s, d)，代表目標為找出s到d的最短路徑，接著會輸入一個整數n，代表接下來會有n條有向且有權重的邊，接下來的n行輸入，每行會有兩格字元(a,b)及一個整數(m)，代表有一條邊由a連至b且權重為m。

Output

輸出s至d的最短路徑長

在實作中我寫了兩種class：Node以及Graph，分別對應資料結構的節點與圖

class Node(): 🡪創建Node class

    def \_\_init\_\_(self, alpha): 🡪建構子輸入alpha(英文字母)

        self.alpha = alpha 🡪設定self.alpha為alpha

        self.cost = 999

🡪設定self.cost為999(無限)，用來儲存s到該Node的cost

        self.pointed\_node\_list = []

🡪設定self.pointed\_node\_list為[], 用來儲存可以連接到的所有Node

        self.pointed\_node\_cost\_list = []

🡪設定self. pointed\_node\_cost\_list為[], 用來儲存所有路徑的cost

class Graph(): 🡪創建Graph class

    def \_\_init\_\_(self): 🡪建構子不須輸入設定

        self.node\_list = [] 🡪設定self.node\_list為[]

    def append\_node(self, node):

🡪該方法輸入node，append到self.node\_list中

        self.node\_list.append(node)

    def append\_path(self, begin\_alpha, end\_alpha, cost):

🡪該方法輸入開始與結束字母以及cost，會從開始字母創建一條路徑指向結束字母，並且記錄cost

        for i in range(len(self.node\_list)):

🡪從Graph的node\_list中做歷遍查找

            if self.node\_list[i].alpha == begin\_alpha:

🡪若核對到開始字母，則提出該node準備做動作

                begin\_node = self.node\_list[i]

            if self.node\_list[i].alpha == end\_alpha:

🡪若核對到結束字母，則提出該node準備做動作

                end\_node = self.node\_list[i]

        if end\_node not in begin\_node.pointed\_node\_list:

🡪若結束字母不在開始字母的指向node list裡面的限制，防止輸入路徑重複

            begin\_node.pointed\_node\_list.append(end\_node)

🡪將結束的node加入到開始的node的pointed\_node\_list

            begin\_node.pointed\_node\_cost\_list.append(cost)

🡪將cost加入到開始的node的pointed\_node\_cost\_list

🡪pointed\_node\_list以及pointed\_node\_cost\_list的長度及對應位置應相同

    def dijkstra\_walk(self, source\_alpha, target\_alpha):

🡪該方法執行dijkstra算法，輸入源字母與終字母，回傳終字母的最低cost

        walked\_node\_list = [] 🡪創建記錄走過的node的list

        BFS\_node\_queue = [] 🡪廣度優先搜尋的queue實作

        for node in self.node\_list:                 # init s cost = 0

🡪對Graph中的node\_list作歷遍查找

            if node.alpha == source\_alpha: 🡪若有node是源字母

                node.cost = 0 🡪初始化源字母的cost為0

                BFS\_node\_queue.append(node) 🡪push 源node進queue

        while walked\_node\_list != self.node\_list:   # walk around all nodes

🡪若還有node沒有走過歷遍到就不停止

            poped\_node = BFS\_node\_queue.pop(0) 🡪pop node queue

            walked\_node\_list.append(poped\_node) 🡪紀錄該node被歷遍到了

            for pointed\_node in poped\_node.pointed\_node\_list: # renew pointed\_node cost

🡪對於被pop的node，歷遍它有指向的其他node

                new\_cost = poped\_node.cost + poped\_node.pointed\_node\_cost\_list[poped\_node.pointed\_node\_list.index(pointed\_node)]

🡪新的cost為pop的node的cost加上對應路徑的cost

                #print('=>', pointed\_node.alpha, new\_cost)

                if new\_cost < pointed\_node.cost:

🡪若新的cost的確有比被指向的node的cost要低

                    #print('cost:', new\_cost, pointed\_node.cost)

                    pointed\_node.cost = new\_cost

🡪則更新之

                if pointed\_node not in walked\_node\_list and pointed\_node not in BFS\_node\_queue:

🡪而且如果被指向的node沒有被走過且不在queue中，就append

                    #print('input:', pointed\_node.alpha)

                    BFS\_node\_queue.append(pointed\_node)

        for node in self.node\_list: 🡪最後一次歷遍node\_list

            if node.alpha == target\_alpha: 🡪尋找終節點

                return node.cost 🡪並回傳終節點的cost

以下為主程式實作：

if \_\_name\_\_ == '\_\_main\_\_': 🡪主程式

    input\_str = input() 🡪接受輸入

    source\_alpha, target\_alpha = input\_str[0], input\_str[2]

🡪分離出源字母與終字母

    inputed\_alpha\_list = [] 🡪用以紀錄輸入過的字母

    graph = Graph() 🡪創建Graph實例

    path\_num = int(input()) 🡪接受輸入

    for i in range(path\_num): 🡪重複這麼多次

        input\_str\_list = input().split(' ') 🡪分離輸入字串

        begin\_alpha, end\_alpha, cost = input\_str\_list[0], input\_str\_list[1], int(input\_str\_list[2])

🡪取得每一行輸入的開始字母、結束字母與cost

        if begin\_alpha not in inputed\_alpha\_list: 🡪若開始字母沒有輸入過

            inputed\_alpha\_list.append(begin\_alpha)🡪紀錄開始字母

            graph.append\_node(Node(begin\_alpha))

🡪呼叫graph物件的append\_node，傳送開始字母的node實例

        if end\_alpha not in inputed\_alpha\_list: 🡪若結束字母沒有輸入過

            inputed\_alpha\_list.append(end\_alpha) 🡪紀錄結束字母

            graph.append\_node(Node(end\_alpha))

🡪呼叫graph物件的append\_node，傳送結束字母的node實例

        graph.append\_path(begin\_alpha, end\_alpha, cost)

🡪呼叫graph物件的append\_path，建立開始到結束node的路徑

    shortest\_cost = graph.dijkstra\_walk(source\_alpha, target\_alpha)

🡪執行graph物件的dijkstra\_walk，回傳終字母的最低cost

    print(shortest\_cost) 🡪印出終字母的最低cost