以下是dijkstra演算法的步驟

1. 至起始點找尋尚未拜訪的相鄰結點

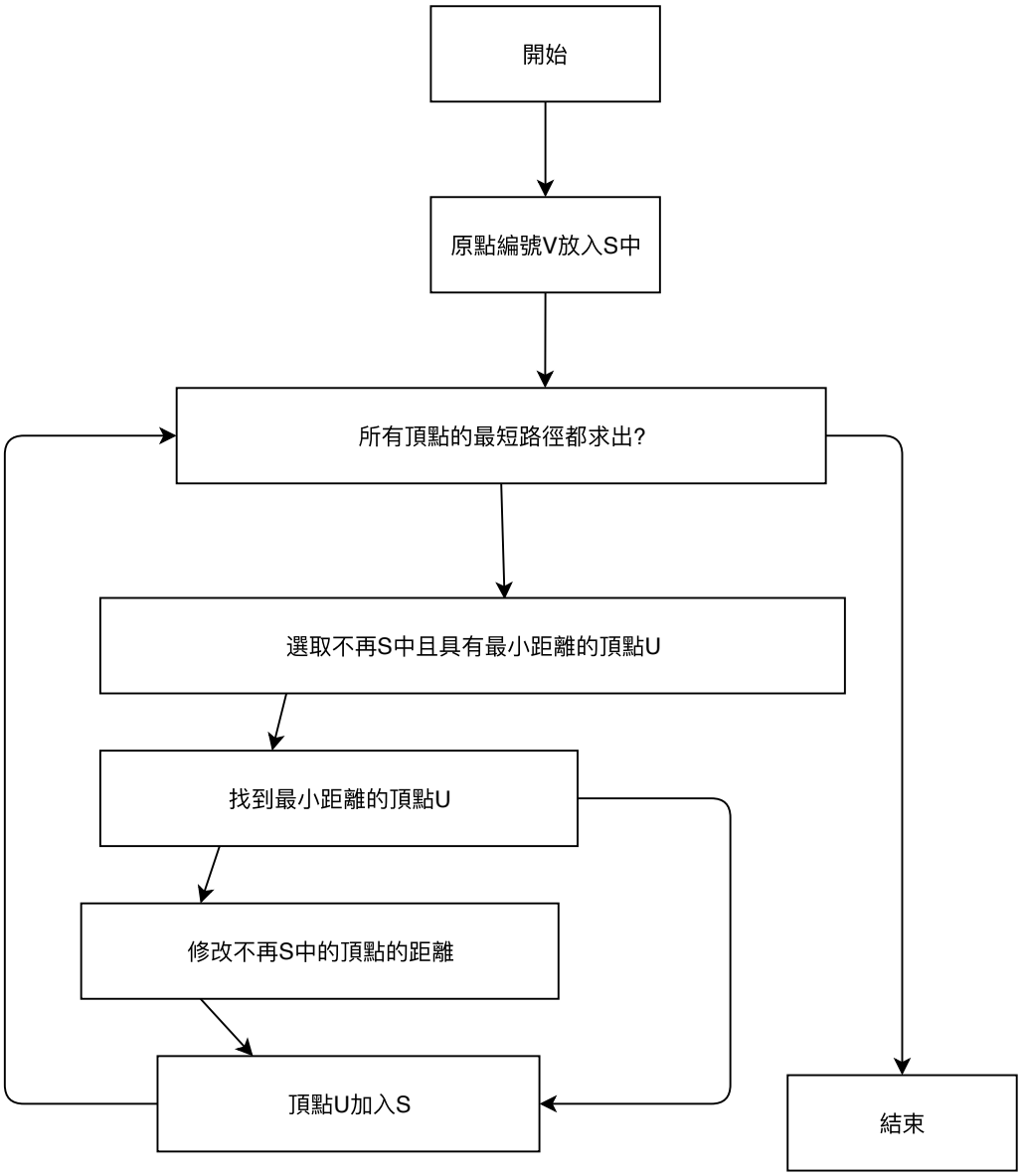
2. 更新最短路徑表

3. 找尋目前未拜訪的最短路徑結點，將此結點設為起始點，並設為已拜訪

4.重複第一步，直到所有結點皆為已拜訪

此處拜訪的定義為---已得到最短路徑

Dijkstra整體的架構



了解架構後我開始寫程式碼

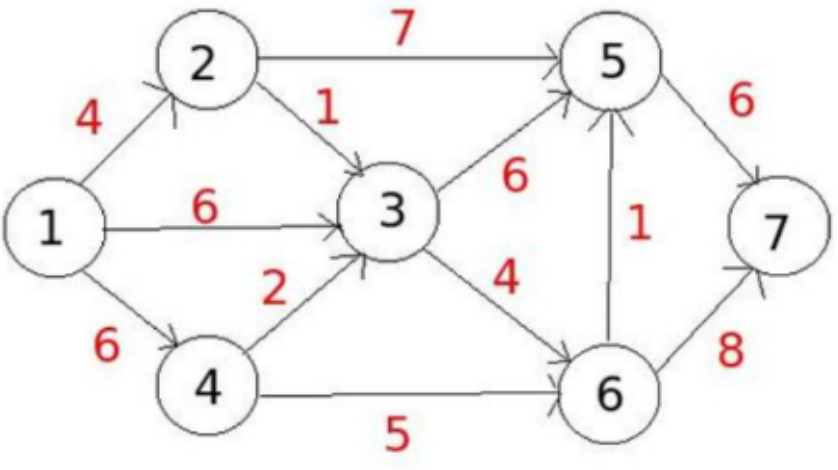
```
while unvisited:
    neighbors = self.graph_matrix[curr]
    for i, x in enumerate(neighbors):
        if x == 0 or i in visited: continue
        values[i] = min(values[i], values[curr] + x)
```

當x是0或是已被拜訪過，則繼續找出自身的質或是相加比自身值還要小

```
min_val = max(values)

if not unvisited: break
for node in unvisited:
    if values[node] < min_val:
        next_curr = node
        min_val = values[node]
curr = next_curr
```

如果以拜訪就停止



#1
起始點：1 （1已拜訪）

最短路徑表

D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0	4	6	6	-	-	-
P[1]	P[2]	P[3]	P[4]	P[5]	P[6]	P[7]
1	1	1	1	1	1	1

#2
起始點：2 （1 2已拜訪）

D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0	4	5	6	11	-	-
P[1]	P[2]	P[3]	P[4]	P[5]	P[6]	P[7]
1	1	2	1	2	1	1

#3
起始點：3 （1 2 3已拜訪）

D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0	4	5	6	11	9	-
P[1]	P[2]	P[3]	P[4]	P[5]	P[6]	P[7]
1	1	2	1	2	3	1

#4
起始點：4 （1 2 3 4已拜訪）

D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0	4	5	6	11	9	-
P[1]	P[2]	P[3]	P[4]	P[5]	P[6]	P[7]
1	1	2	1	2	3	1

#5
起始點：6 （1 2 3 4 6已拜訪）

D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0	4	5	6	10	9	17
P[1]	P[2]	P[3]	P[4]	P[5]	P[6]	P[7]
1	1	2	1	6	3	6

#6
起始點：5 （1 2 3 4 5 6已拜訪）

D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0	4	5	6	10	9	16
P[1]	P[2]	P[3]	P[4]	P[5]	P[6]	P[7]
1	1	2	1	6	3	5

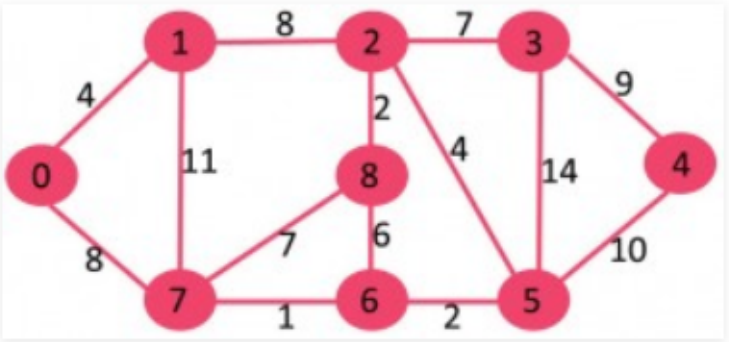
#7
起始點：7 （1 2 3 4 5 6 7已拜訪）

D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
0	4	5	6	10	9	16
P[1]	P[2]	P[3]	P[4]	P[5]	P[6]	P[7]
1	1	2	1	6	3	5

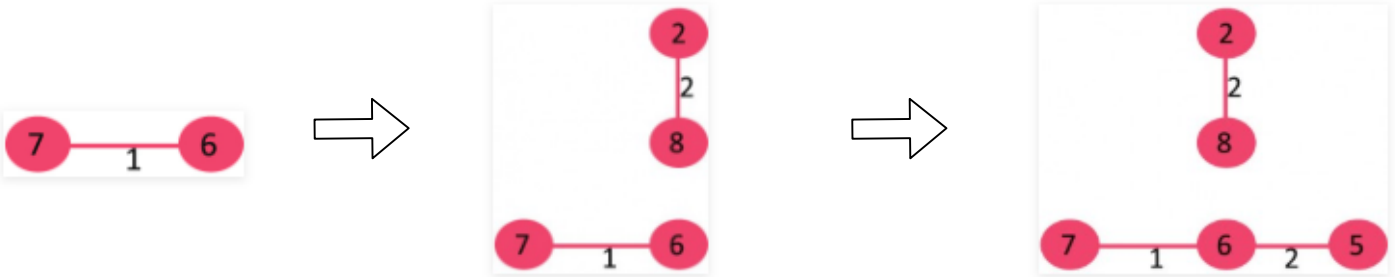
)

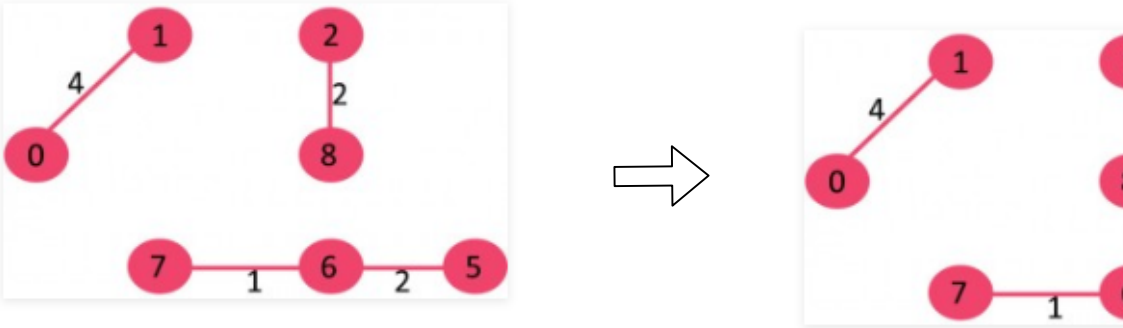
Kruskal是另一個計算最小生成樹的演算法，其演算法原理如下。首先，將每個頂點放入其自身的資料集合中。然後，按照權值的升序來選擇邊。當選擇每條邊時，判斷定義邊的頂點是否在不同的資料集中。如果是，將此邊插入最小生成樹的集合中，同時，將集合中包含每個頂點的聯合體取出，如果不是，就移動到下一條邊。重複這個過程直到所有的邊都探查過。

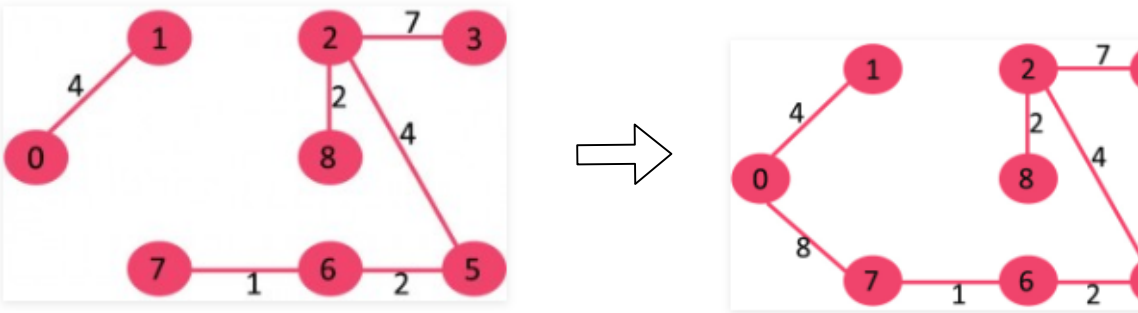
原本長這樣



圖解







參考資料

https://www.youtube.com/watch?v=pVfj6mxhdMw
https://www.runoob.com/python3/python3-set.html
http://www.csie.ntnu.edu.tw/~u91029/SpanningTree.html
https://www.geeksforgeeks.org/detect-cycle-undirected-graph/
https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/