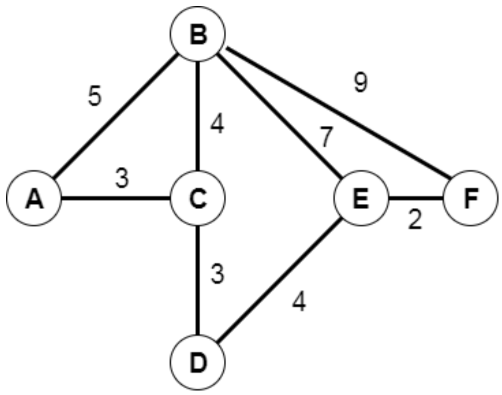
一、 Dijkstra 與 Kruskal 流程圖

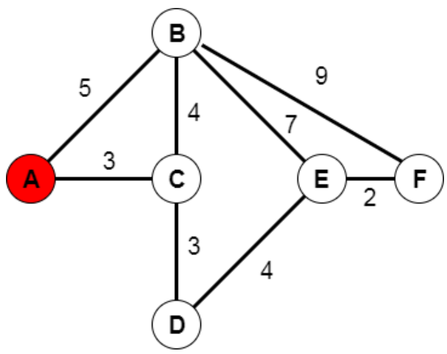
Dijkstra



	A	B	C	D	E	F
A	0	5	3	∞	∞	∞
B	5	0	4	∞	7	9
C	3	4	0	3	∞	∞
D	∞	∞	3	0	4	∞
E	∞	7	∞	4	0	2
F	∞	9	∞	∞	2	0

將左圖轉換成右圖的矩陣，如果是走到自己距離為 0；無法走到的點為無限。

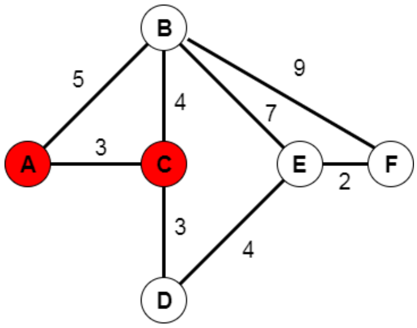
<Step1>



	A	B	C	D	E	F
A	0	5	3	∞	∞	∞

1. 選擇起始點 A。
2. 標記 A 點到各點的距離。
3. 將 A 欄位標記為「已計算出最小距離」的記號，起始點到 A 點的最小距離為 0。

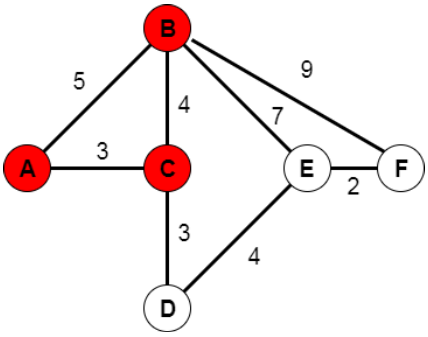
<Step2>



	A	B	C	D	E	F
A	0	5	3	∞	∞	∞
A C	0	5	3	6	∞	∞

1. 從 A 列中除了已被標記為「已計算出最小距離」的點外，選擇離起始點最近的點 C。
2. 新增 A C 列，更新 A C 列的數值。如果加入 C 點後，從起始點 A 到其他點的距離相對沒有加入 C 點還短則更新數值。其中 D 點距離起始點的距離縮短為 6
3. 將 C 欄位標記為「已計算出最小距離」的記號，起始點到 C 點的最小距離為 3。

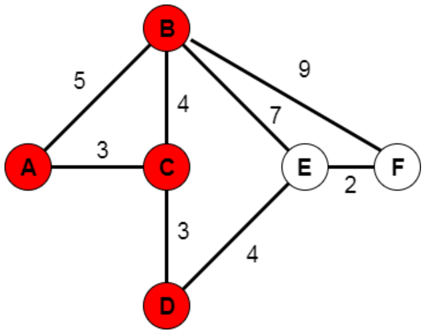
<Step3>



	A	B	C	D	E	F
A	0	5	3	∞	∞	∞
A C	0	5	3	6	∞	∞
A C B	0	5	3	6	12	14

1. 從 A C 列中除了已被標記為「已計算出最小距離」的點外，選擇離起始點最近的點 B。
2. 新增 A C B 列，更新 A C B 列的數值。如果加入 B 點後，從起始點 A 到其他點的距離相對沒有加入 B 點還短則更新數值。其中 E、F 點距離起始點的距離縮短為 12、14。
3. 將 B 欄位標記為「已計算出最小距離」的記號，起始點到 B 點的最小距離為 5。

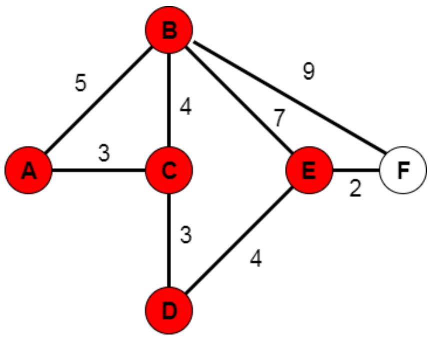
<Step4>



	A	B	C	D	E	F
A	0	5	3	∞	∞	∞
A C	0	5	3	6	∞	∞
A C B	0	5	3	6	12	14
A C B D	0	5	3	6	10	14

1. 從 A C B 列中除了已被標記為「已計算出最小距離」的點外，選擇離起始點最近的點 D。
2. 新增 A C B D 列，更新 A C B D 列的數值。如果加入 D 點後，從起始點 A 到其他點的距離相對沒有加入 D 點還短則更新數值。其中 E 點距離起始點的距離縮短為 10。
3. 將 D 欄位標記為「已計算出最小距離」的記號，起始點到 D 點的最小距離為 6。

<Step5>

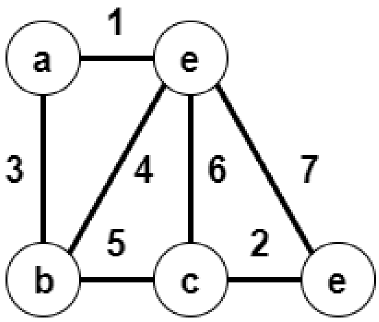


	A	B	C	D	E	F
A	0	5	3	∞	∞	∞
A C	0	5	3	6	∞	∞
A C B	0	5	3	6	12	14
A C B D	0	5	3	6	10	14
A C B D E	0	5	3	6	10	12

1. 從 A C B D 列中除了已被標記為「已計算出最小距離」的點外，選擇離起始點最近的點 E。
2. 新增 A C B D E 列，更新 A C B D E 列的數值。如果加入 E 點後，從起始點 A 到其他點的距離相對沒有加入 E 點還短則更新數值。其中 F 點距離起始點的距離縮短為 12。
3. 將 E 欄位標記為「已計算出最小距離」的記號，起始點到 E 點的最小距離為 10。

結束

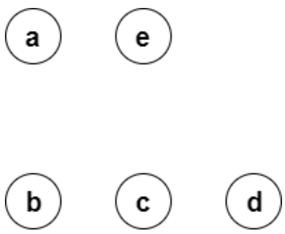
Kruskal



<Step1>

1. 將各邊線依權值大小由小到大排列。
2. 建立 disjoint set。在初始情況下，每個 vertex 的 root 都是自己本身。
3. 在初始情況下，MST 的每個 vertex 都未相連。

Minimum Spanning Tree



權值	S	D
1	a	e
2	c	d
3	a	b
4	b	e
5	b	c
6	e	c
7	e	d

Disjoint Set

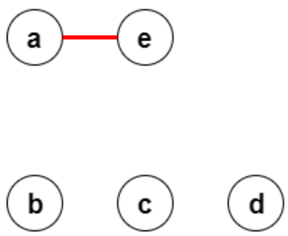
vertex	a	b	c	d	e
root	a	b	c	d	e

<Step2>

1. 從權值最低的邊依序選起，其中一點為 start，另一點為 destination
2. 分別檢查 start 和 destination 的 root，如果 root 相同代表有迴路，不選；如果不同則在 minimum spanning tree 中連接該兩點
3. 在 disjoint set 中將 destination 的 root 改成 start 的 root。
4. 將其他與 start 原本的 root 相同的 vertex 的 root 改成 start 新的 root。
5. 直到在 minimum spanning tree 中連接了|V|−1 條邊線結束

<Step2-1>

Minimum Spanning Tree



權值	S	D
1	a	e
2	c	d
3	a	b
4	b	e
5	b	c
6	e	c
7	e	d

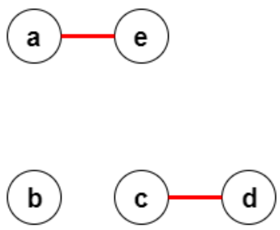
Disjoint Set

vertex	a	b	c	d	e
root	a	b	c	d	e

vertex	a	b	c	d	e
root	a	b	c	d	a

<Step2-2>

Minimum Spanning Tree



權值	S	D
1	a	e
2	c	d
3	a	b
4	b	e
5	b	c
6	e	c
7	e	d

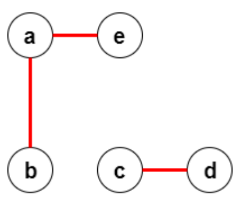
Disjoint Set

vertex	a	b	c	d	e
root	a	b	c	d	a

vertex	a	b	c	d	e
root	a	b	c	c	a

<Step2-3>

Minimum Spanning Tree



權值	S	D
1	a	e
2	c	d
3	a	b
4	b	e
5	b	c
6	e	c
7	e	d

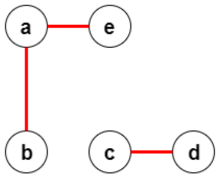
Disjoint Set

vertex	a	b	c	d	e
root	a	b	c	c	a

vertex	a	b	c	d	e
root	a	a	c	c	a

<Step2-4>

Minimum Spanning Tree



權值	S	D
1	a	e
2	c	d
3	a	b
4	b	e
5	b	c
6	e	c
7	e	d

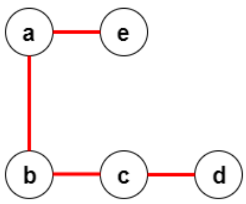
Disjoint Set

vertex	a	b	c	d	e
root	a	a	c	c	a

vertex	a	b	c	d	e
root	a	a	c	c	a

<Step2-5>

Minimum Spanning Tree



權值	S	D
1	a	e
2	c	d
3	a	b
4	b	e
5	b	c
6	e	c
7	e	d

Disjoint Set

vertex	a	b	c	d	e
root	a	a	c	c	a

vertex	a	b	c	d	e
root	a	a	a	a	a

二、 程式碼學習歷程

改良版 Dijkstra:

```python
class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[None for column in range(vertices)] for row in range(vertices)]
        self.index = []
        self.outcome = dict()
        self.finished = [False] * vertices

    def addEdge(self,u,v,w):
        if u not in self.index:
            self.index.append(u)
            self.outcome[str(u)] = None
            self.graph[self.index.index(u)][self.index.index(u)] = 0

        if v not in self.index:
            self.index.append(v)
            self.outcome[str(v)] = None
            self.graph[self.index.index(v)][self.index.index(v)] = 0

        self.graph[self.index.index(u)][self.index.index(v)] = w
        self.graph[self.index.index(v)][self.index.index(u)] = w
```

```python
    def Dijkstra(self, s, initialize = True):
        if initialize == True:
            s_index = self.index.index(s)
            for i in self.index:
                d_index = self.index.index(i)
                cost = self.graph[s_index][d_index]
                self.outcome[str(i)] = cost
            self.finished[s_index] = True

        else:
            base = self.outcome[str(s)]
            s_index = self.index.index(s)
            for i in self.index:
                d_index = self.index.index(i)
                if s != i and self.graph[s_index][d_index] != None:
                    d = base + self.graph[s_index][d_index]
                    if self.outcome[str(i)] == None and self.finished[d_index] == False:
                        self.outcome[str(i)] = d
                    elif self.outcome[str(i)] != None and d < self.outcome[str(i)]:
                        self.outcome[str(i)] = d
            self.finished[s_index] = True

        if False in self.finished:
            cost = 999
            for i in self.outcome.items():
                s_index = self.index.index(int(i[0]))
                if self.finished[s_index] == False and i[1] != None:
                    if i[1] < cost:
                        target = int(i[0])
                        cost = i[1]
            return self.Dijkstra(target, initialize = False)

        else:
            return self.outcome
```

圖中每個點的值本身沒有意義，可以是 0, 1, 2, 3,、A, B, C, D 或 37, 44,12, 13。為了因應點的值不規則的情況，增設 self.index 這個陣列，在進行 self.addEdge 時，把每個點 append 進 index 這個陣列中，每個值的索引值為 index.index()，範圍為 range(0,self.V)。這樣就可以把每值在 Index 這個陣列中的索引值對應到 Self.graph 這個 matrix 中，之後要呼叫每個值時就直接呼叫該值在 index 這個陣列中的索引值，並拿這個索引值找到該值在 Self.graph 的位置。

程式碼說明 – **Kruskal Algorithm:**

```
1  class Graph():
2  
3      def __init__(self, vertices):
4          self.V = vertices
5          self.graph2 = []
6          self.outcome2 = dict()
7          self.dj = dict()
8  
9      def addEdge(self,u,v,w):
10         self.graph2.append((w, u, v))
11 
12         if u not in self.dj.keys():
13             self.dj[u] = u
14 
15         if v not in self.dj.keys():
16             self.dj[v] = v
17 
18     def Kruskal(self):
19         self.graph2 = sorted(self.graph2, key = lambda item:item[0])
20 
21         for edge in self.graph2:
22             w, s, d = edge[0], edge[1], edge[2]
23 
24             if self.dj[d] != self.dj[s]:
25                 tmp = self.dj[d]
26                 self.dj[d] = self.dj[s]
27 
28                 for point in self.dj.keys():
29                     if self.dj[point] == tmp:
30                         self.dj[point] = self.dj[s]
31 
32                 key = str(s) + '-' + str(d)
33                 self.outcome2[key] = w
34 
35             if len(self.outcome2) == self.V - 1:
36                 break
37 
38         return self.outcome2
39 
```

第 5~7 行。self.graph2 用來存放權重(w)、出發點(s)、目的點(d)。self.outcome 用來存放最後 spanning tree 的邊。Self.dj 為 disjoint set。

第 24~26 行。分別檢查 start 和 destination 的 root，如果 root 相同代表有迴路，不選；如果不同則在 minimum spanning tree 中連接該兩點。在 disjoint set 中將 destination 的 root 改成 start 的 root。

第 28~30。將其他與 start 原本的 root 相同的 vertex 的 root 改成 start 新的 root。

程式碼說明 – Dijkstra Algorithm:

```
1 class Graph():
2 
3     def __init__(self, vertices):
4         self.V = vertices
5 
6         self.graph = [[None for column in range(vertices)] for row in range(vertices)]
7         self.outcome = dict()
8         self.finished = [False] * vertices
10    def Dijkstra(self, s, initialize = True):
11        if initialize == True:
12
13            for i, j in zip(range(self.V), self.graph[s]):
14                self.outcome[str(i)] = j
15
16            self.finished[s] = True
17
18        else:
19
20            base = self.outcome[str(s)]
21
22            for i in range(self.V):
23
24                if s != i and self.graph[s][i] != 0:
25                    d = base + self.graph[s][i]
26
27                    if self.outcome[str(i)] == 0 and self.finished[i] == False:
28                        self.outcome[str(i)] = d
29
30                    elif self.outcome[str(i)] != 0 and d < self.outcome[str(i)]:
31                        self.outcome[str(i)] = d
32
33            self.finished[s] = True
34
35        if False in self.finished:
36            order = sorted(self.outcome.items(), key = lambda item:item[1])
37            target = None
38
39            for i in order:
40
41                if self.finished[int(i[0])] == False and i[1] != 0:
42                    target = int(i[0])
43                    break
44
45            return self.Dijkstra(target, initialize = False)
46
47        else:
48            return self.outcome
```

第 6~8 行。self.graph 用來存放 adjacent matrix；self.outcome 用來存放起始點到各點的最短距離；self.finished 用來標記該點使否已計算玩最短距離。

第 11~16 行。計算起始點到各點的最短距離。其中 initialize 為 True，之後的遞迴程式的 initialize 設為 False。

第 20~43 行。除了已被標記為「已計算出最小距離」的點外，選擇離起始點最近的點選擇離起始點最近的點。更新起始點到各點的最短距離，從起始點到各點的距離相對沒有加入新的點還短時則更新數值，標記選擇過的為「已計算出最小距離」的記號。

第 45 行。如仍有點的最短距離還未被計算完則進行遞迴。

三、 Dijkstra 與 Kruskal 原理說明

Shortest Path

想要知道從高雄到台南，如果開車的話，通常會利用查詢的交通網路來取得: 最短路徑或者走哪一條路最符合經濟效益。諾以圖形網路來思考，就是任意兩個頂點之間的最短路徑或最少花費。

Dijkstra Algorithm

該方法用於計算由某個頂點到其他頂點的最短路徑。

D[k] = A[F, k], (k = 1, N)

S = {F}, V = {1, 2, ……, N}

- A[F, k] 為頂點 F 到 k 的距離。
- D 為一個 N 維陣列用來存放某一頂點到所有頂點的最短距離。
- F 表示起始頂點，V 是網路中所有頂點的集合。
- S 也是頂點的集合。

重複執行以下兩個步驟，直到 V-S 是空集合為止。

1. 從 V – S 集合中找到一個頂點 x，使得 D(x)為最小值，並把 x 放入 S 集合中。
2. 依 D[k] = min(D[k], D[x] + A[x, k])來調整陣列 D 得值。

該方法具有下列幾項特性:

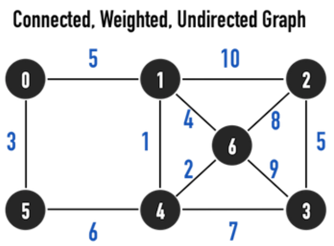
1. 如果 u 是目前所找到最短路徑之下一個節點，則 u 必屬於 V-S 集合中最小花費成本的邊。
2. 若 u 被選中，將 u 加入 S 集合中，則產生目前由起始點到 u 的最短路徑。

Minimum Spanning Tree (MST)

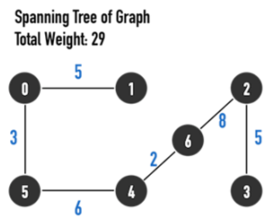
Spanning Tree 需滿足三個條件:

1. 連結所有 Graph 中的 vertex (點)的樹。
2. 沒有 cycle，代表一個 vertex 只能有一個 root。
3. 若 Graph 有 V 個 vertex，Spanning Tree 只有|V|−1 條 edge。

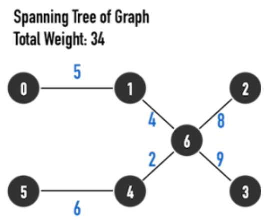
如圖一，考慮一個 connected、weighted 的 undirected graph，相同的 vertex 會有不同的 Spanning Tree，如圖二、三皆為其 Spanning Tree 的寫法。



圖一



圖二



圖三

由於 Graph 具有 weight，因此，不同的 Spanning Tree，可能有不同的 weight 總和，而其中，具有最小 weight 總和的樹，稱為 Minimum Spanning Tree (MST)。MST 有很多實際應用。將網路頂點看做城市，邊看做連線城市的通訊網，邊的權重看做連線城市的通訊線路的成本，根據最小生成樹建立的通訊網就是這些城市之間成本最低的通訊網。其中如果有相同權值的邊線時，MST 並非唯一；如邊線的權值均不同，則 MST 唯一。

Kruskal Algorithm

Kruskal Algorithm 為建立 MST 的其中一種方法。該演算法將各邊線依權值大小由小到大排列，從權值最低的邊線開始架構 MST，依序拿最小成本的邊來搭建 MST，如果加入的邊線會造成迴路則捨棄不用，直到加入了 |V|−1 條邊線為止。

四、 參考資料

Dijkstra 與 Kruskal 原理說明:

1. http://alrightchiu.github.io/SecondRound/minimum-spanning-treeintrojian-jie.html
2. http://alrightchiu.github.io/SecondRound/minimum-spanning-treekruskals-algorithm.html
3. 課堂 Minimum Spanning Tree 講義
4. 課堂 Shortest Path 講義
5. 李淑馨，資料結構使用 PYTHON，深石出版，2019

程式碼:

1. 課堂 Minimum Spanning Tree 講義
2. 課堂 Shortest Path 講義