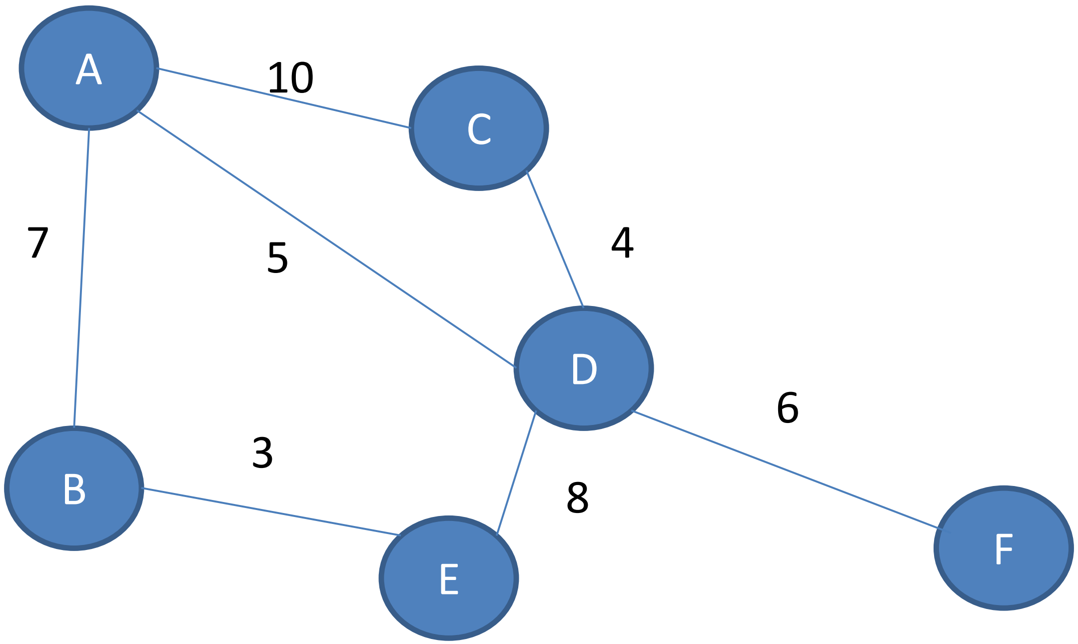
北一女中
資訊選手培訓營

最短路徑
Shortest Path

By Nan 2012.08. 13

什麼是最短路徑？

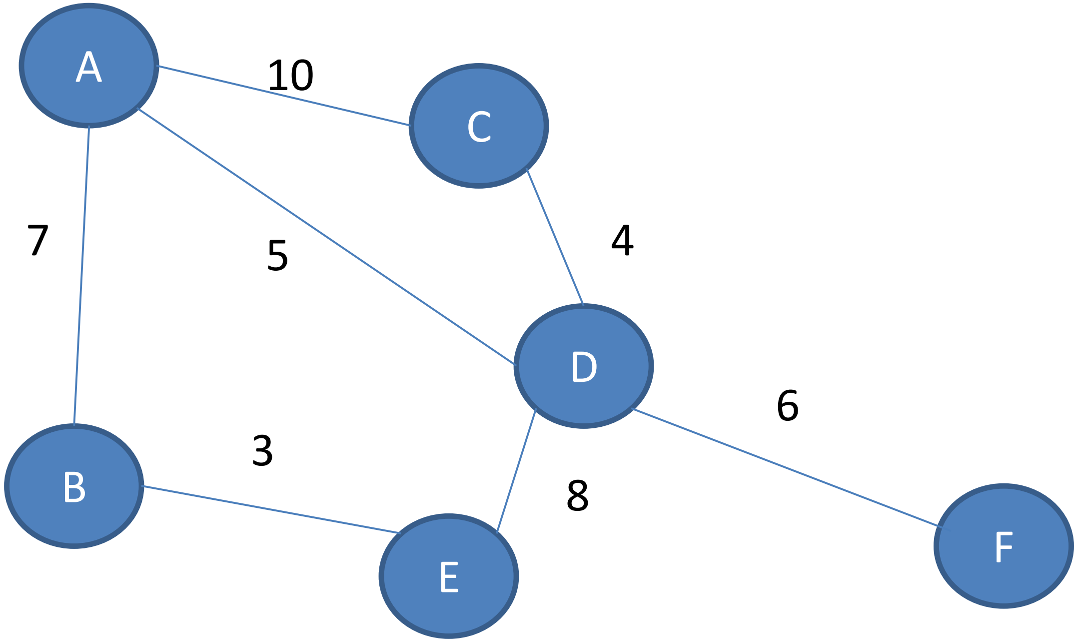


Outline

- Single Source Shortest Path 單源最短路徑
  - Dijkstra’s Algorithm
  - Bellman-Ford Algorithm
- All-pairs Shortest Path 全點對最短路徑
  - Floyd-Warshall Algorithm

找起點到所有點的最短距離(和路徑)

SINGLE SOURCE SHORTEST PATH

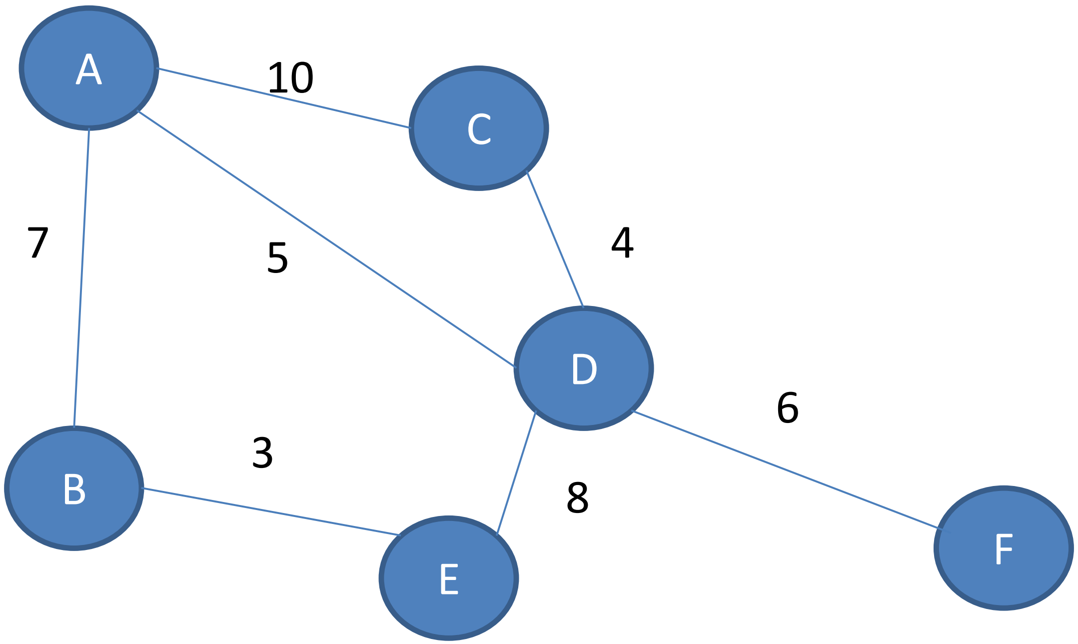

A
B
C
D
E
F
10
7
5
4
3
8
6

使用條件：圖上沒有權值為負之邊

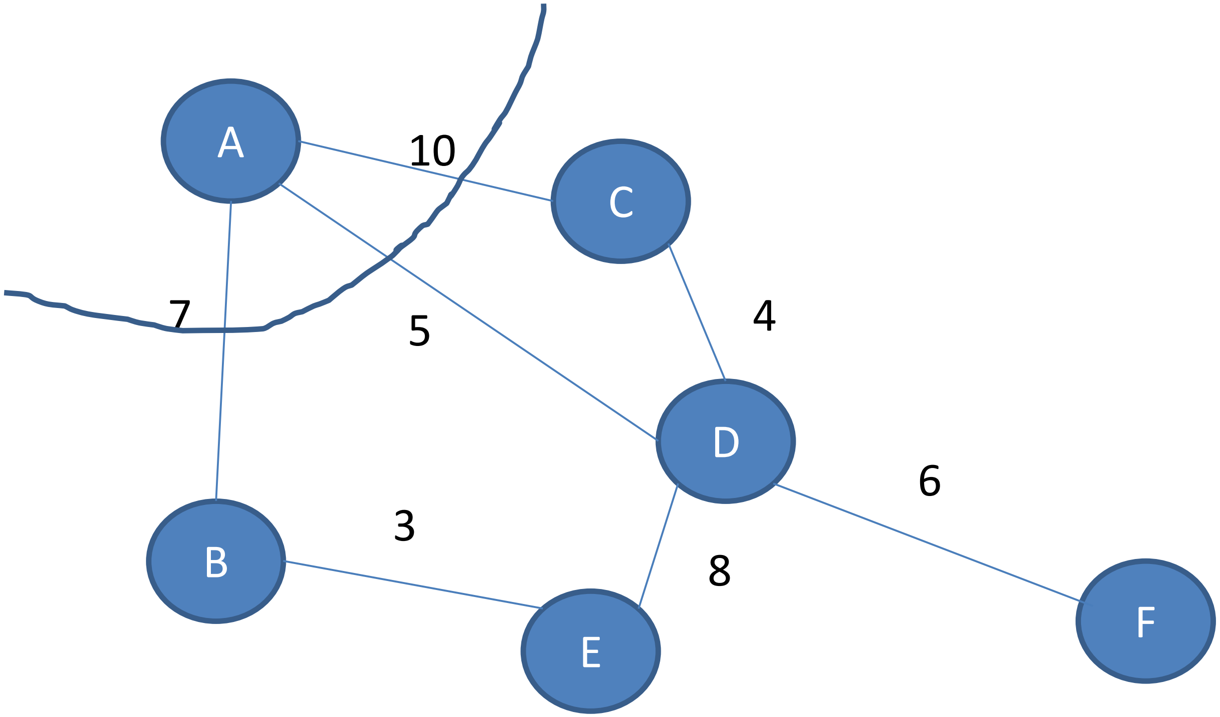
DIJKSTRA’ S ALGORITHM

A	B	C	D	E	F
0	inf	inf	inf	inf	inf

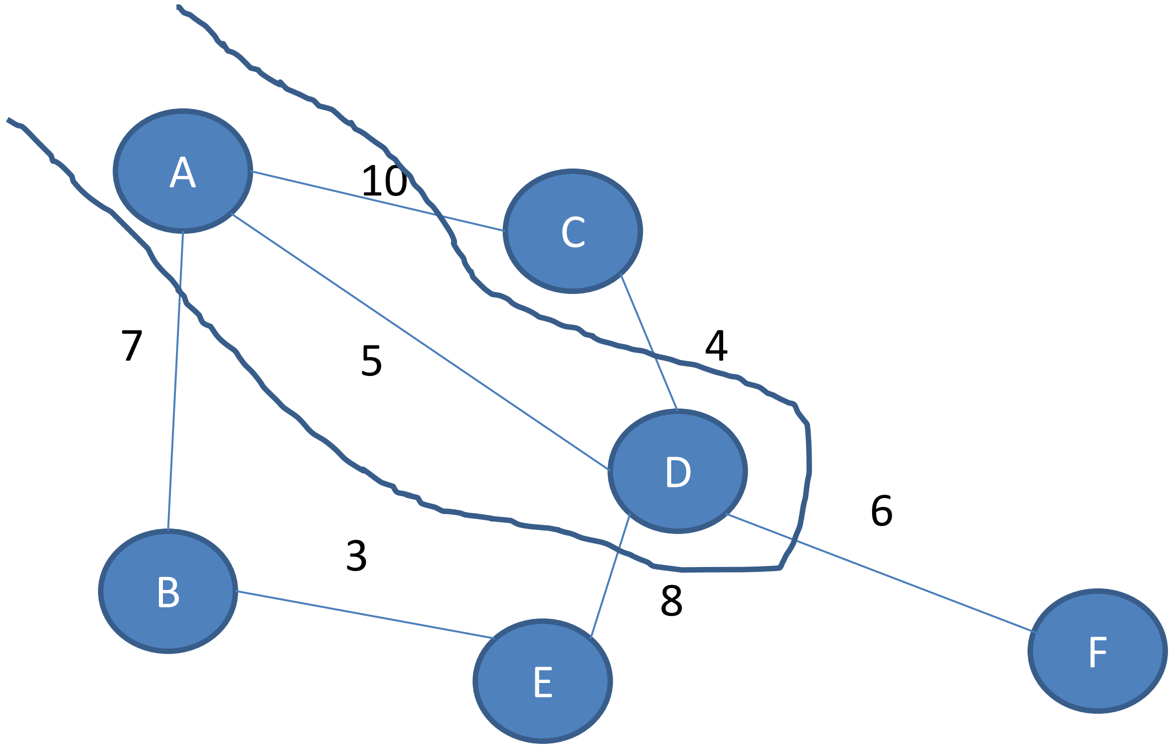
「正確聯盟」--從起點到他的點的距離已經確定的點



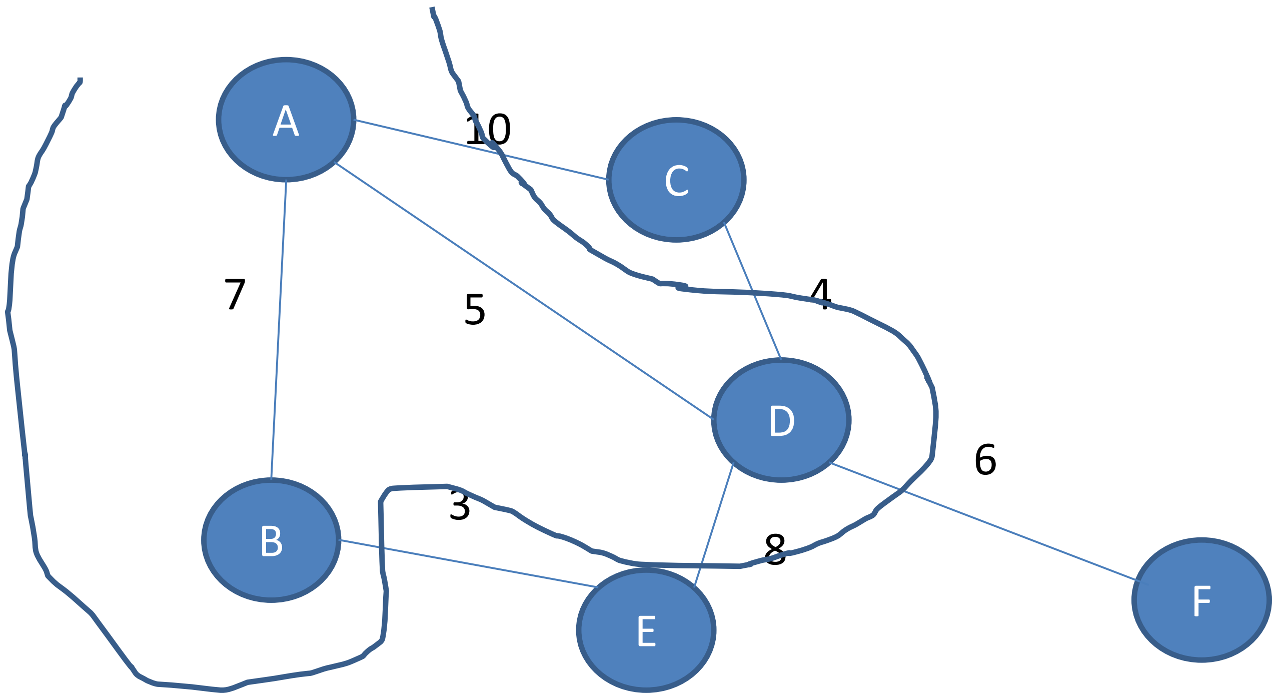
A	B	C	D	E	F
0	7	10	5	inf	inf



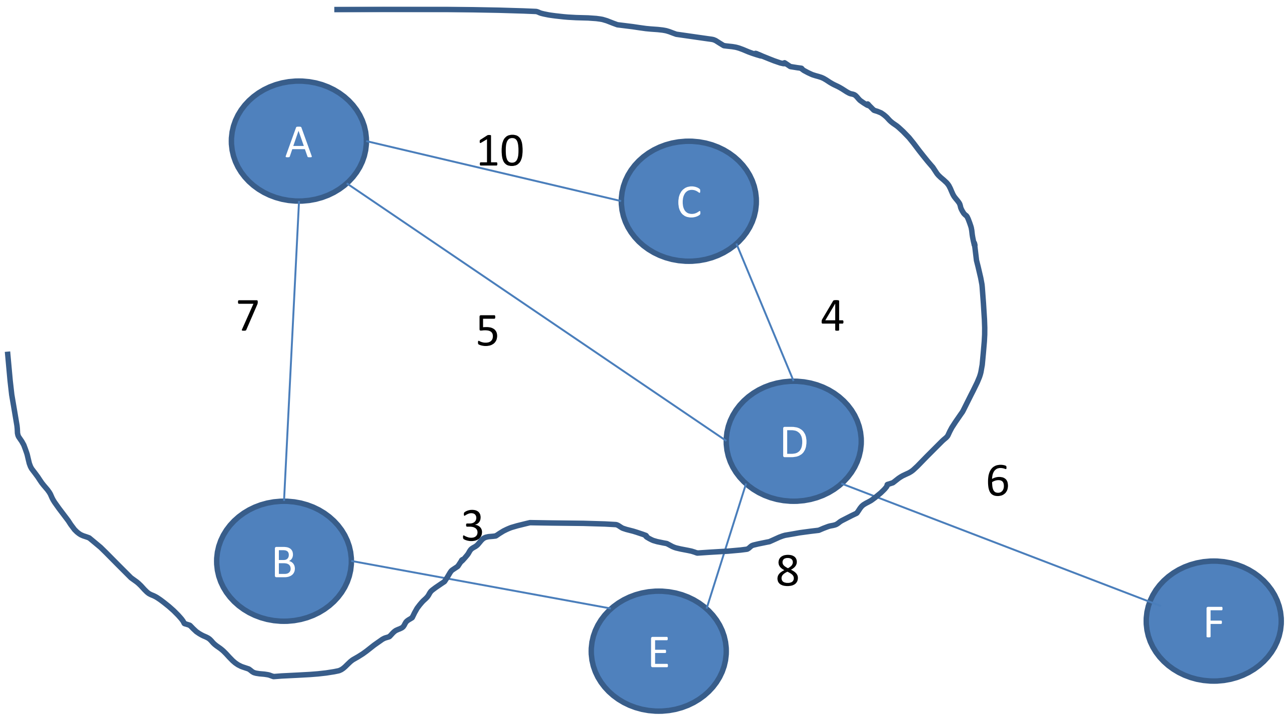
A	B	C	D	E	F
0	7	9	5	13	11



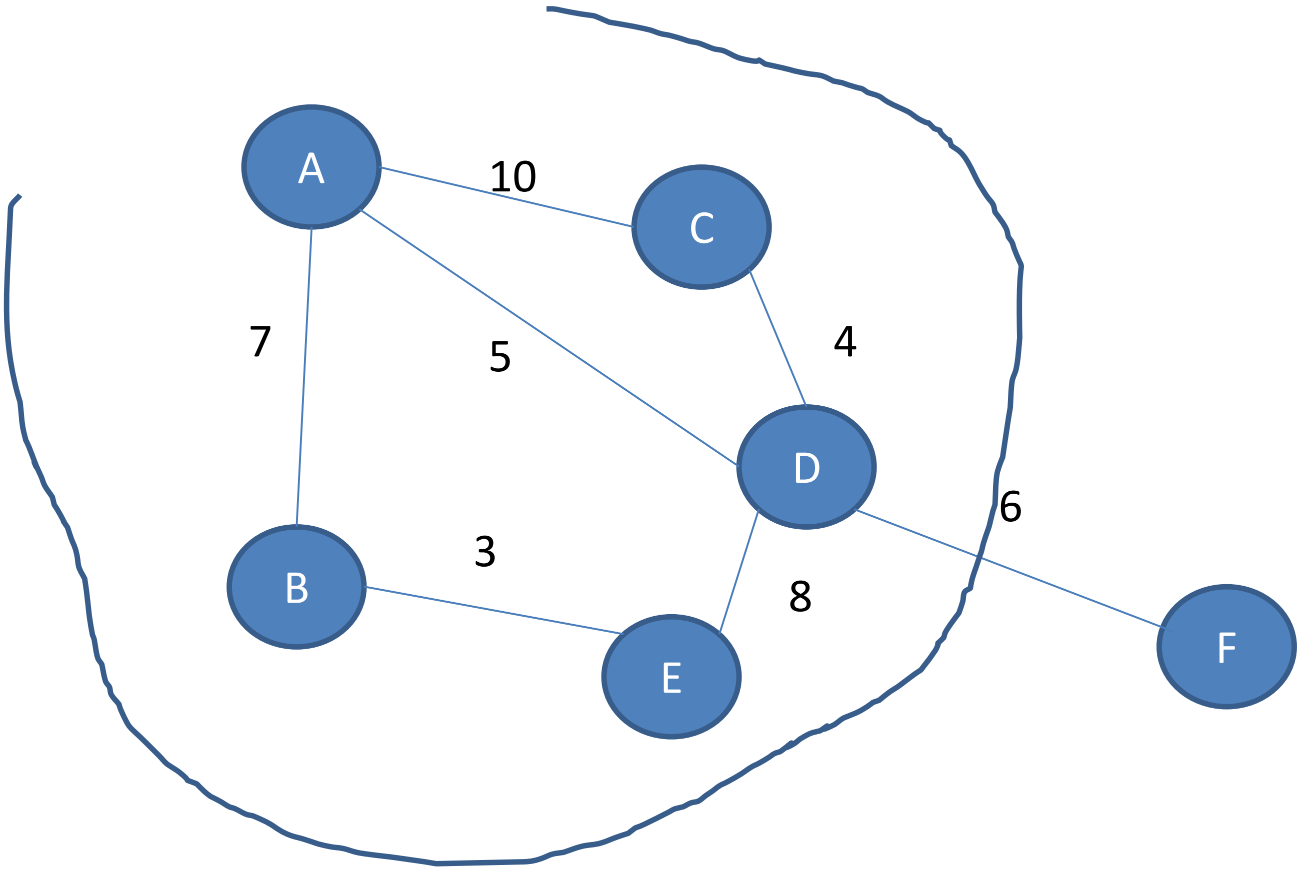
A	B	C	D	E	F
0	7	9	5	10	11



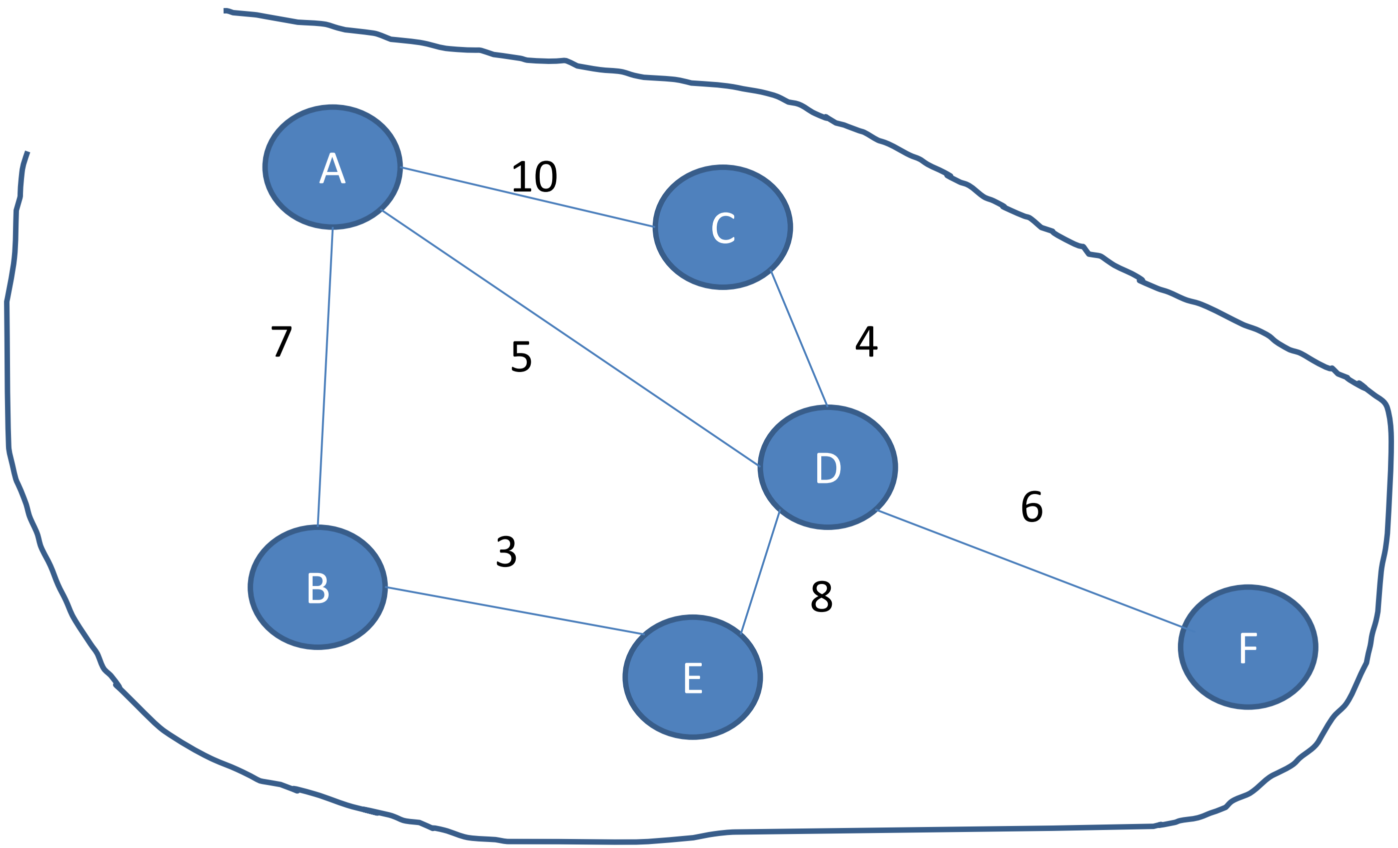
A	B	C	D	E	F
0	7	9	5	10	11



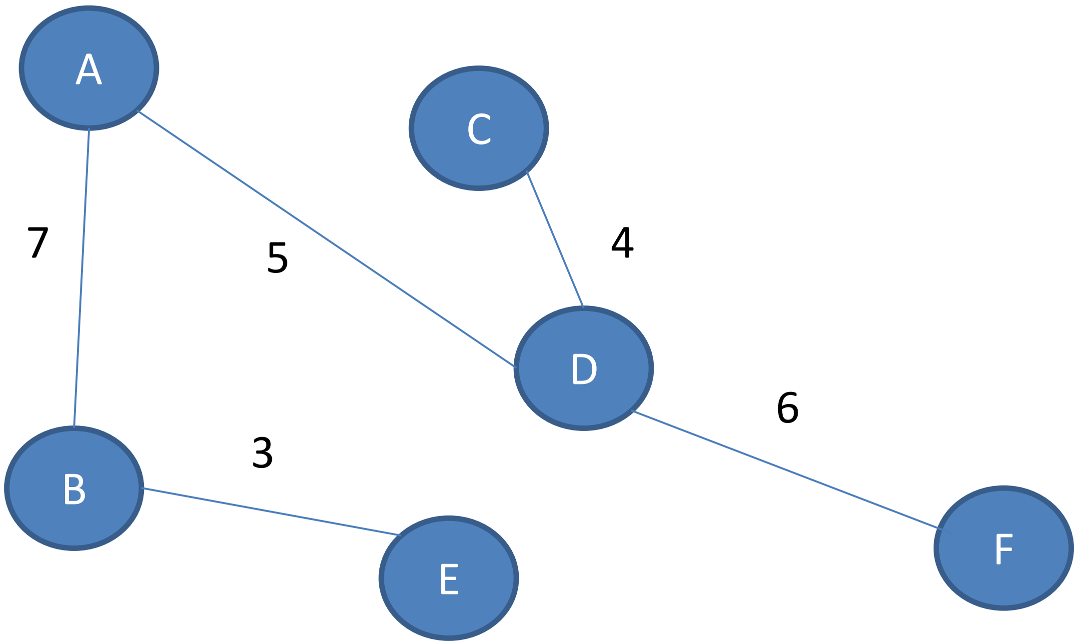
A	B	C	D	E	F
0	7	9	5	10	11



A	B	C	D	E	F
0	7	9	5	10	11



A	B	C	D	E	F
0	7	9	5	10	11



其他一些要注意的

- 要記得是找所有可能加進正確聯盟的點最近的，不是找「新加進的點距離誰最近」。
- 要找出整條路徑的方法：另開一個表格，紀錄他是透過誰更新成現在的值的。

```
#include <string.h>         // 為了用memset這個function所以要include這個
#define INF 2147483647      // 用int的最大值做為無限大
int graph[N][N];            // 假設我們有N個點。這裡存的是邊(i,j)的距離(無向邊)
                            // 沒有邊時的距離就是INF

int dist[N];                // 記錄目前要把第i個點加入正確聯盟所需要的距離
int last[N];                // 記錄第i個點是透過誰加入了正確聯盟(等於是存在edge(last[i], i))
int choosed[N];             // 記錄是否已經加入了正確聯盟
int fin_cnt;                // 記錄已經加入正確聯盟的點的個數

void init(){                // 初始化
    // memset會把整塊記憶體空間都填上零，有歸零作用(但不能用來歸成除了0和-1之外的其他值)。
    memset(choosed, 0, sizeof(choosed));
    // last = -1代表自己就是root，一開始所有點都是自己的root
    memset(last, -1, sizeof(last));

    // 以idx=0的點作為root開始看距離
    dist[0] = 0;
    choosed[0] = 1;
    int i;
    for ( i = 1 ; i < N ; i++ ){
        dist[i] = graph[0][i];   // 如果有邊dist就會是該條邊，反之則會是INF
        if ( dist[i] != INF )
            last[i] = 0;
    }

    fin_cnt = 1;                 // 一開始只有一個點在正確聯盟裡
}
```

```
void dijkstra(){
    int min;                                    // 用來存這一輪找到的距離最小值
    int min_idx;                                // 用來存這一輪找到距離最小的是哪個點
    int i;
    while ( fin_cnt < N ) {                     // 如果小於N代表還沒找完
        min = INF;                              // 初始化成INF，用來找最小值
        min_idx = -1;                           // 初始化成-1，之後用來判別有沒有找到新的可用的點
        for ( i = 1 ; i < N ; i++ ){            // 跑過所有點，找最小值
            if ( choosed[i] == 1 )              // 已經在正確聯盟裡就不考慮
                continue;
            if ( dist[i] < min ){
                min_idx = i;
                min = dist[i];
            }
        }
        if ( min_idx == -1 ) break;             // 如果沒找到代表此圖找不到下一個可更新的點

        choosed[min_idx] = 1;                   // 標記min_idx這個點進入了正確聯盟
         fin_cnt++;                             // fin_cnt增加一，代表多了一個點已經確定

        // 看看還沒有被選的點，有沒有點能夠透過min_idx這個點而更近的
        for ( i = 1 ; i < N ; i++ ){
            if ( choosed[min_idx] == 1 ) continue;   // 被選過的就跳過
            if ( ((dist[min_idx] + graph[min_idx][i]) < dist[i] ){    // 有更近就更新
                last[i] = min_idx;
                dist[i] = dist[min_idx] + graph[min_idx][i];
            }
        }
    }
}
```

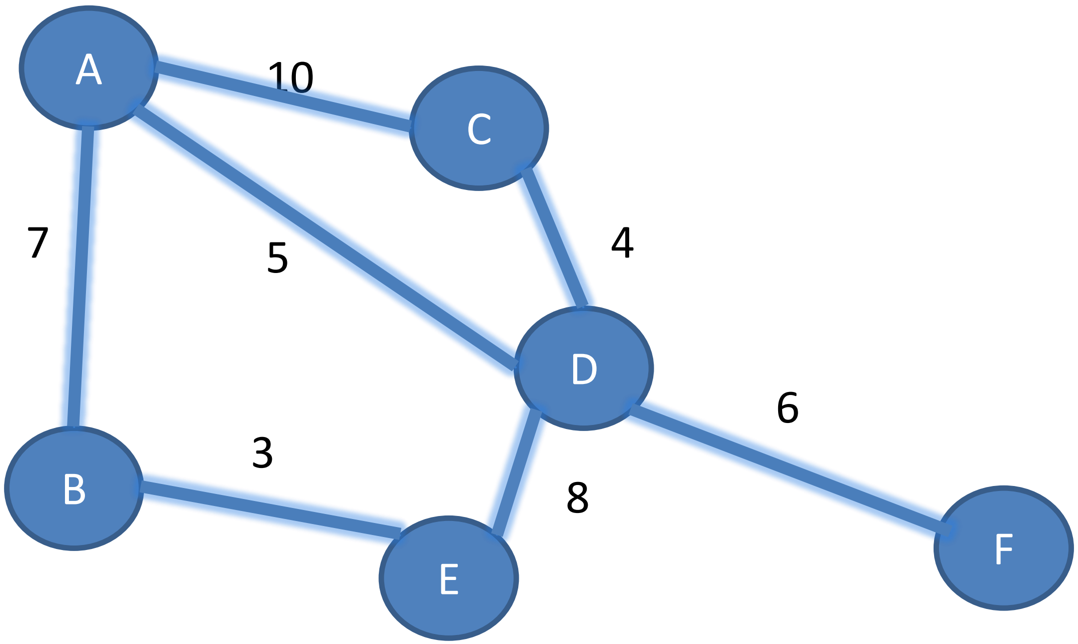
可用在有負邊的情況下，亦可用來檢查負環(negative cycle)

BELLMAN-FORD ALGORITHM

A	B	C	D	E	F
0	7	10	5	inf	inf

每次都枚舉所有的邊(i, j)，兩個方向都看看是否有變短

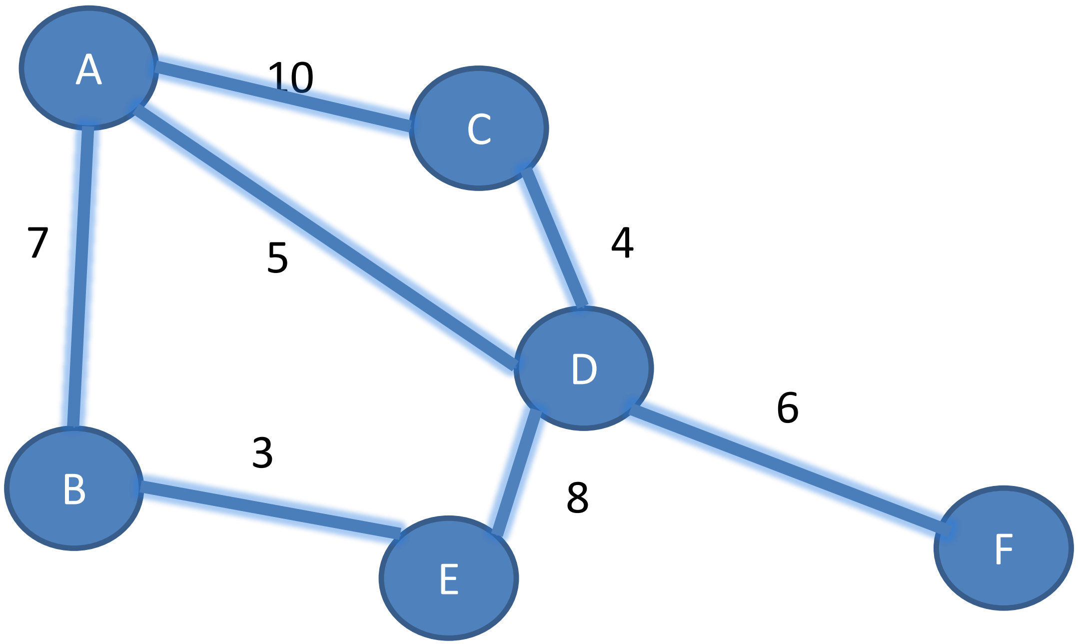
➔也就是看 dist[i] + w(i, j) < dist[j] 或 dist[j] + w(i, j) < dist[i] 是否成立



A	B	C	D	E	F
0	7	9	5	10	11

每次都枚舉所有的邊(i, j)，兩個方向都看看是否有變短

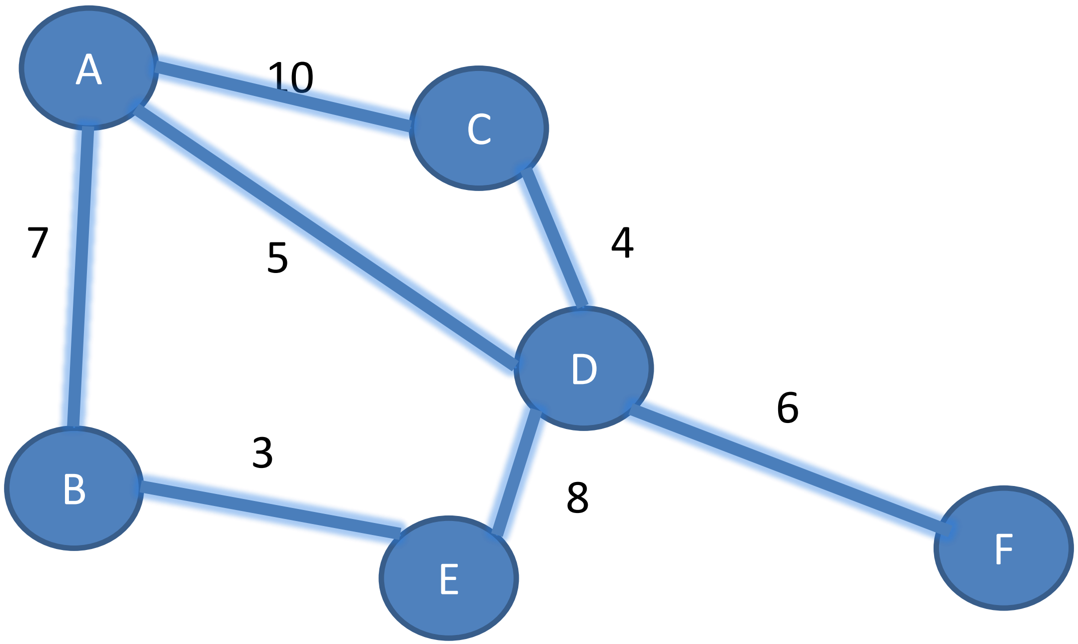
➔也就是看 dist[i] + w(i, j) < dist[j] 或 dist[j] + w(i, j) < dist[i] 是否成立



第i輪代表的意義：最多透過i條邊而走到該點的最近距離

A	B	C	D	E	F
0	7	9	5	10	11

沒有人改變就可以結束了(理論上只有n-1輪，除非有負環)



如果需要知道過程經過的點，
那就要另開表格紀錄最後是透過誰更新成現在的值的

```c
#define INF 2147483647     // 用int的最大值做為無限大
int st[M], ed[M], w[M];    // 假設我們有M條邊。edge(st[i], ed[i])的dist為w[i]
int dist[N];               // 記錄目前從起點開始到第i個點的最近距離
int last[N];               // 記錄第i個點是透過誰而得到現在的最近距離

void init(){               // 初始化
    int i;
    for ( i = 0 ; i < N ; i++ ){
        dist[i] = INF;     // 距離一開始都是無限大
        last[i] = -1;      // 來源初始化
    }
    dist[0] = 0;           // 設起點是編號為0的點
}
void bellman_ford(){
    int i, k,  flag = 1;                        // flag用來記錄有沒有人被改過
    for ( k = 1 ; k < N && flag ; k++ ){        // 最多做N-1回 且 flag必須非0
         flag = 0;                              // 預設是沒有人被改過
          for ( i = 0 ; i < M ; i++ ){           // 跑過所有的邊
            // 先看 st[i]->ed[i]
            if ( dist[st[i]] + w[i] < dist[ed[i]] ){
                  dist[ed[i]]  = dist[st[i]] + w[i];
                  last[ed[i]] = st[i];
                  flag = 1;
            }
            // 再看 ed[i]->st[i]
            if ( dist[ed[i]] + w[i] < dist[st[i]] ){
                  dist[st[i]]  = dist[ed[i]] + w[i];
                  last[st[i]] = ed[i];
                  flag = 1;
            }

        }
    }
}
```

找所有點到所有點的最短距離(和路徑)

ALL PAIRS SHORTEST PATH

有負邊仍可用

FLOYD-WARSHALL ALGORITHM

FW是一個動態規劃的演算法

狀態

$f_k(i, j)$: 從i走到j，中間只能經過編號為1~k的點

最佳子結構

要知道$f_k(i, j)$的最短路徑，必須知道

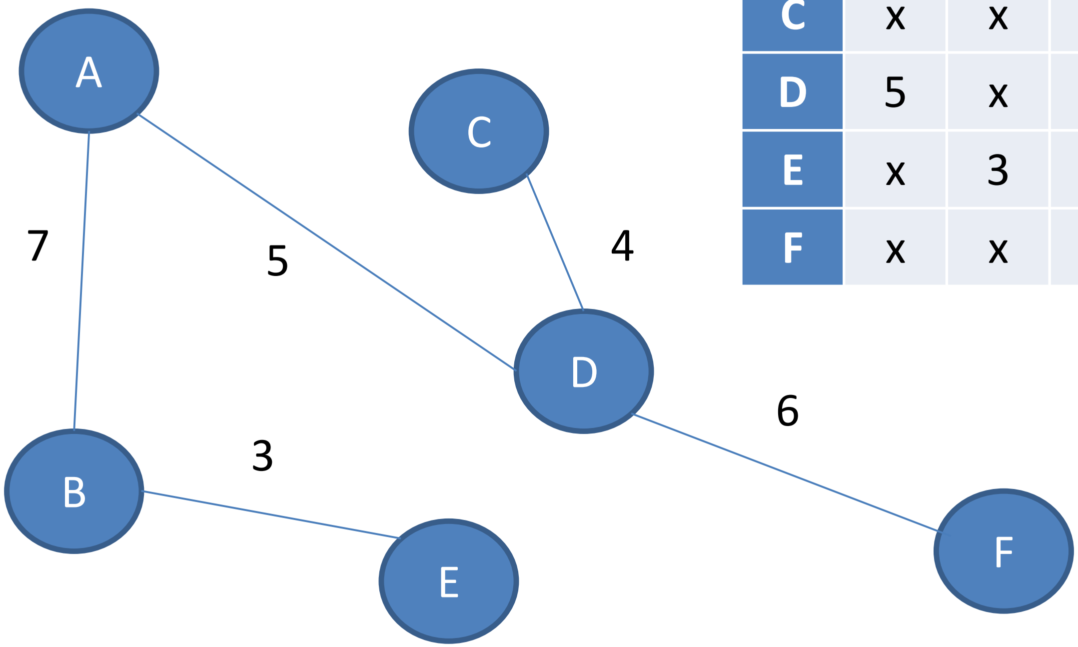
(1) $f_{k-1}(i, j)$的最短路徑 (沒有走到點k的情況)

(2) $f_{k-1}(i, k)$和$f_{k-1}(k, j)$的最短路徑
(加在一起就是經過點k的情況)

遞迴關係

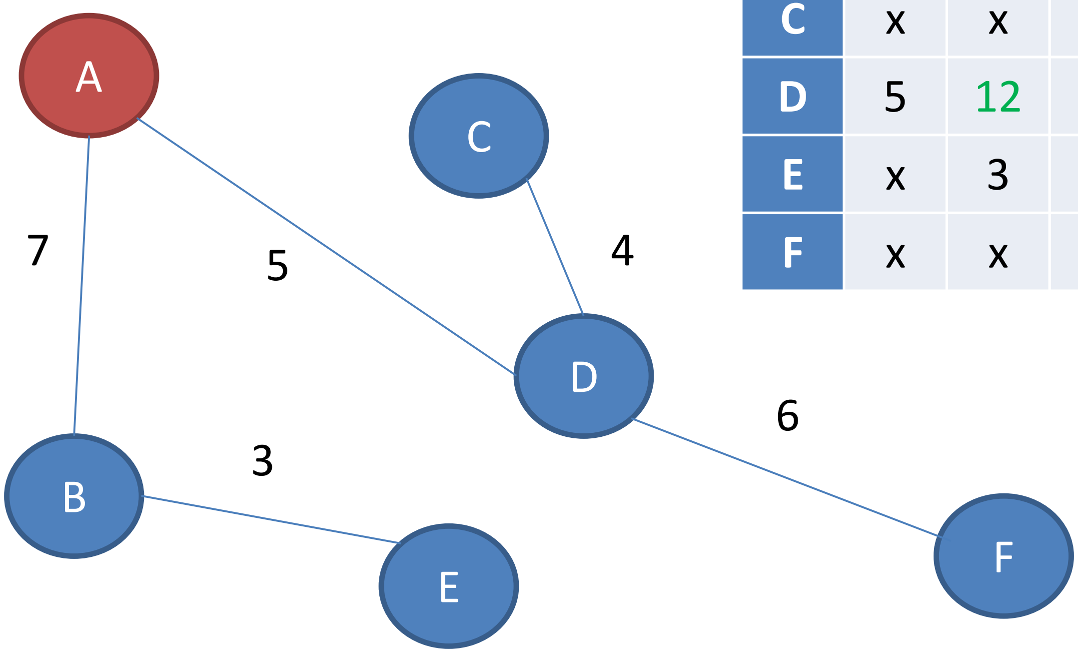
$$\begin{cases} f_k(i,j) = min(f_{k-1}(i,j), f_{k-1}(i,k) + f_{k-1}(k,j)) & \text{for } 0 < k \leq n \\ f_0(i,j) = w(i,j) \end{cases}$$

沒有經過任何其他點就是直接有邊的情況



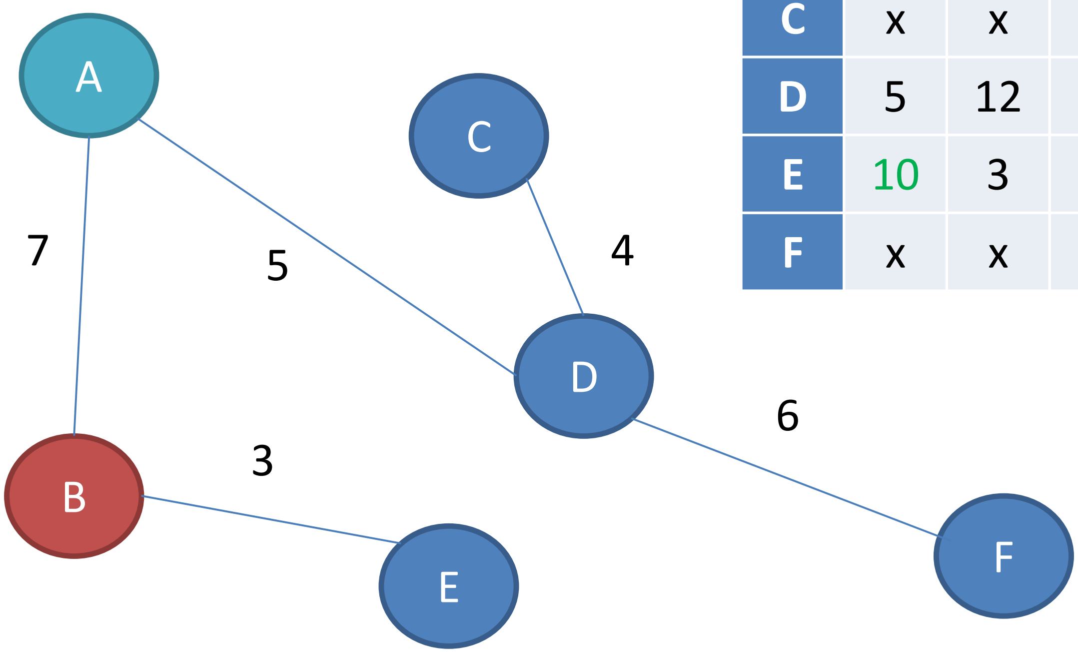
	A	B	C	D	E	F
A	0	7	x	5	x	x
B	7	0	x	x	3	x
C	x	x	0	4	x	x
D	5	x	4	0	x	6
E	x	3	x	x	0	x
F	x	x	x	6	x	0

枚舉(i, A)+(A, j) 去跟(i, j)比



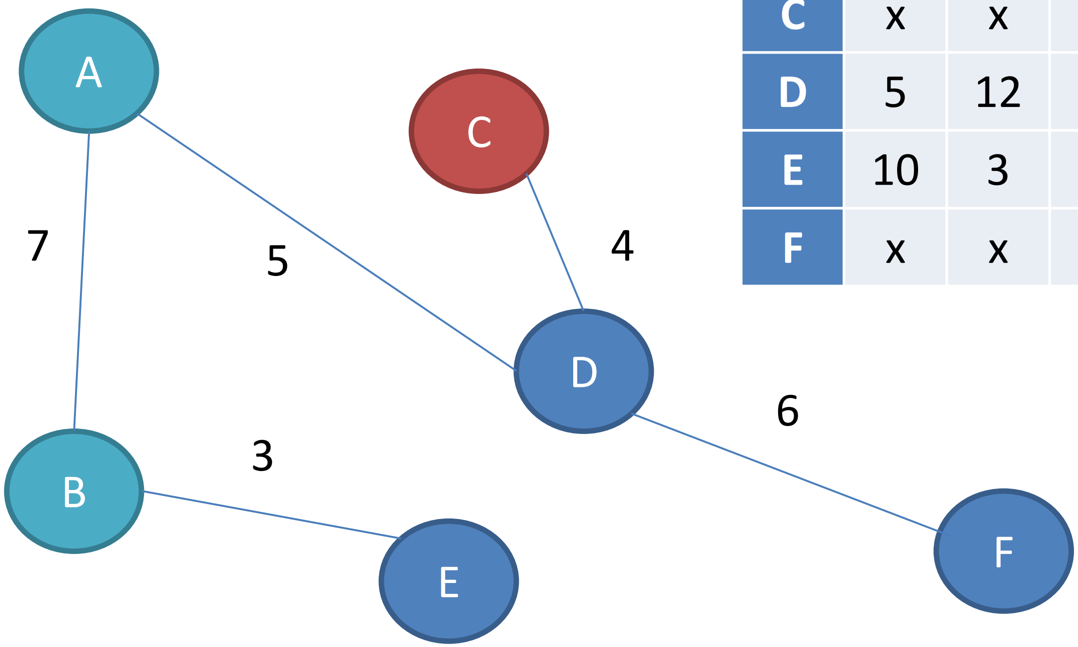
	A	B	C	D	E	F
A	0	7	x	5	x	x
B	7	0	x	12	3	x
C	x	x	0	4	x	x
D	5	12	4	0	x	6
E	x	3	x	x	0	x
F	x	x	x	6	x	0

枚舉(i, B)+(B, j) 去跟(i, j)比



	A	B	C	D	E	F
A	0	7	x	5	10	x
B	7	0	x	12	3	x
C	x	x	0	4	x	x
D	5	12	4	0	15	6
E	10	3	x	15	0	x
F	x	x	x	6	x	0

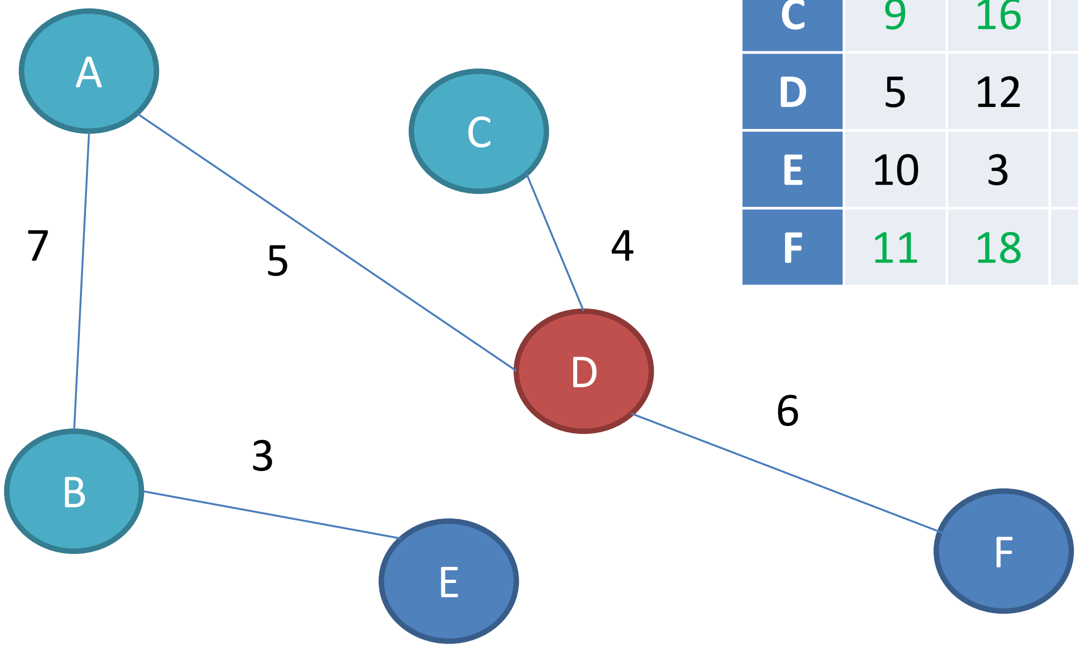
枚舉(i, C)+(C, j) 去跟(i, j)比



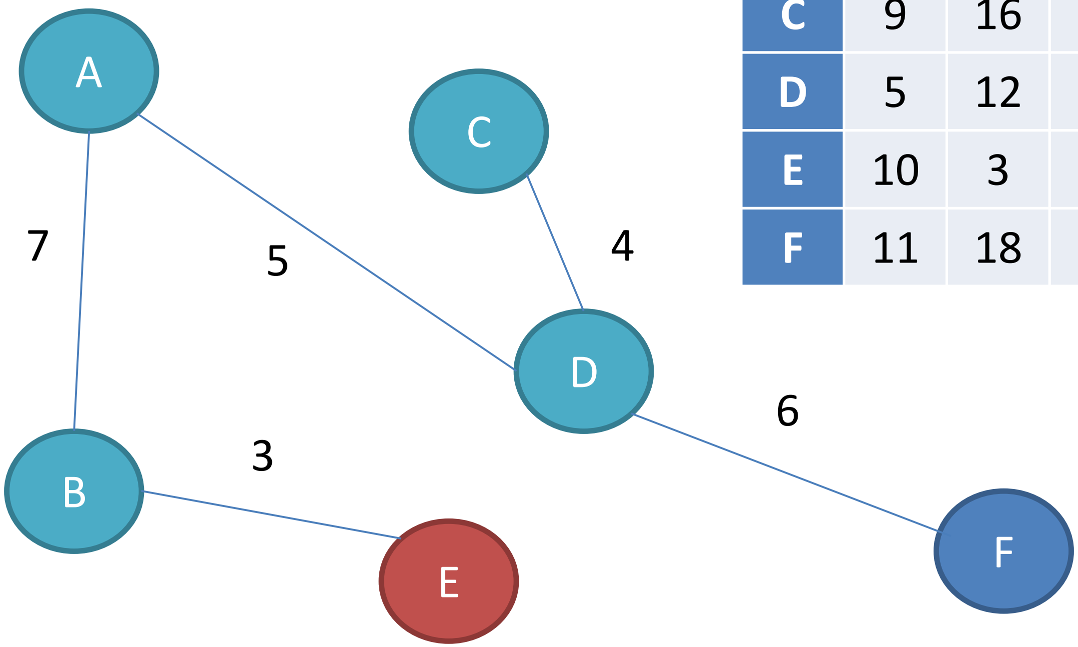
	A	B	C	D	E	F
A	0	7	x	5	10	x
B	7	0	x	12	3	x
C	x	x	0	4	x	x
D	5	12	4	0	15	6
E	10	3	x	15	0	x
F	x	x	x	6	x	0

枚舉(i, D)+(D, j) 去跟(i, j)比

	A	B	C	D	E	F
A	0	7	9	5	10	11
B	7	0	16	12	3	18
C	9	16	0	4	19	10
D	5	12	4	0	15	6
E	10	3	19	15	0	21
F	11	18	10	6	21	0



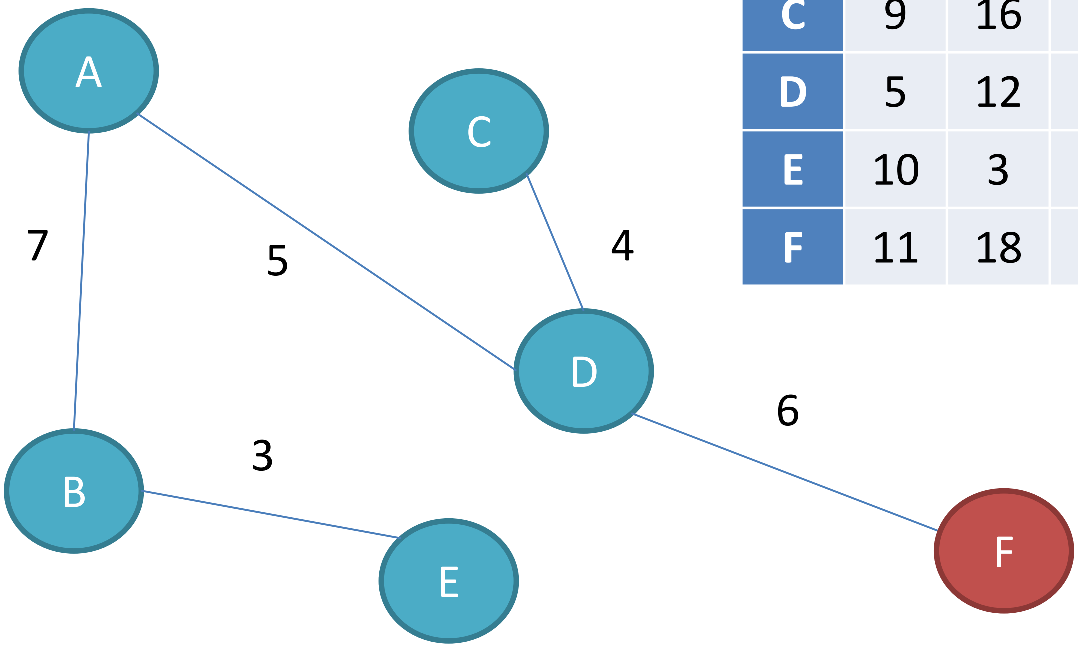
枚舉(i, E)+(E, j) 去跟(i, j)比



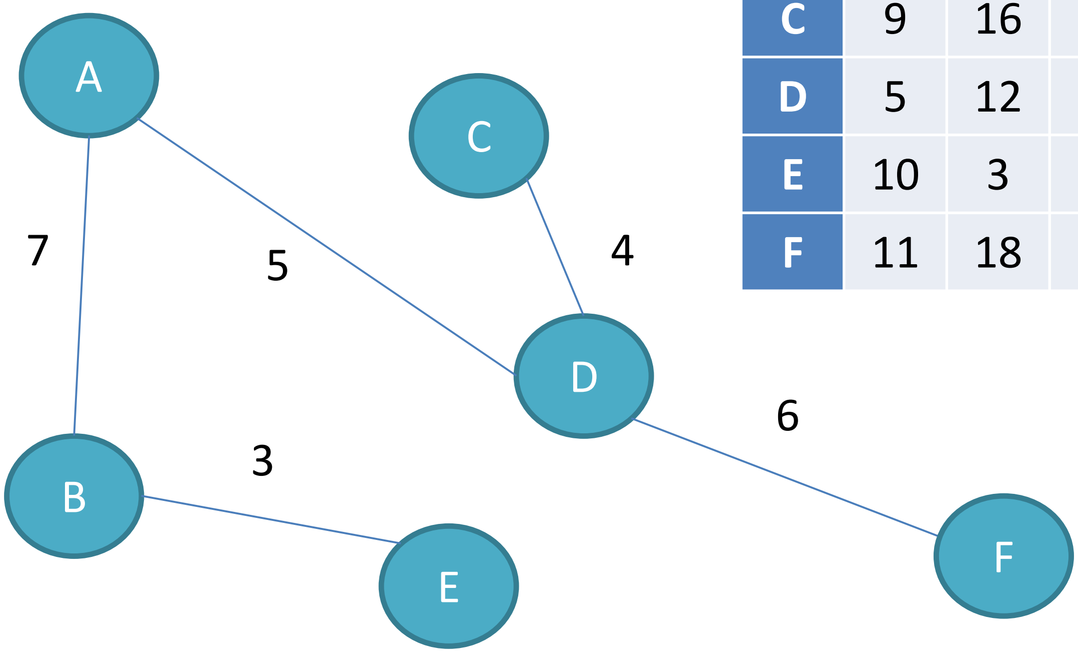
	A	B	C	D	E	F
A	0	7	9	5	10	11
B	7	0	16	12	3	18
C	9	16	0	4	19	10
D	5	12	4	0	15	6
E	10	3	19	15	0	21
F	11	18	10	6	21	0

枚舉(i, F)+(F, j) 去跟(i, j)比

	A	B	C	D	E	F
A	0	7	9	5	10	11
B	7	0	16	12	3	18
C	9	16	0	4	19	10
D	5	12	4	0	15	6
E	10	3	19	15	0	21
F	11	18	10	6	21	0



Done~!



	A	B	C	D	E	F
A	0	7	9	5	10	11
B	7	0	16	12	3	18
C	9	16	0	4	19	10
D	5	12	4	0	15	6
E	10	3	19	15	0	21
F	11	18	10	6	21	0

如果需要知道過程經過的點，
那就要另開表格紀錄最後是透過誰更新成現在的值的

關於開表格的方式

本來狀態$f_k(i, j)$算是有三維，應該要是三維表格的。

➔但因為每輪的k只需要上一輪的k-1時的資訊就好，而表格每格在更新之前就是k-1時的情況，所以可以只用二維。

```c
#define INF 2147483647     // 用int的最大值做為無限大
int graph[N][N];           // 假設我們有N個點。這裡存的是邊(i,j)的距離(無向邊)
                           // 沒有邊時的距離就是INF
int dp[N][N];              // 用來做DP，紀錄距離的表格，初始化會等於graph[i][j]
int last[N][N];            // 記錄目前要把第i個點加入正確聯盟所需要的距離

void init(){               // 初始化
    int i, j;
    for ( i = 0 ; i < N ; i++ ){
        for ( j = 0 ; j < N ; j++ ){
            dp[i][j] = graph[i][j];    // 如果(i, j)有邊就會是該條邊，反之則會是INF
            last[i][j] = -1;           // -1代表沒經過任何點
        }
    }
}

void floyd_warshall(){
    int i, j, k;
    for ( k = 0 ; k < N ; k++ ){
        for ( i = 0 ; i < N ; i++ ){
            for ( j = 0 ; j < N ; j++ ){
                // 起點或終點是當前嘗試的點k   或是起點等於終點   就跳過
                if ( i == j || i == k || j == k ) continue;
                if ( dp[i][k] + dp[k][j] < dp[i][j] ){    // 透過點k有更近就更新
                    dp[i][j] = dp[i][k] + dp[k][j];
                    last[i][j] = k;
                }
            }
        }
    }
}
```

無向邊時可以直接對稱做

```
#define INF 2147483647      // 用int的最大值做為無限大
int graph[N][N];            // 假設我們有N個點。這裡存的是邊(i,j)的距離(無向邊)
                            // 沒有邊時的距離就是INF

int dp[N][N];               // 用來做DP，紀錄距離的表格，初始化會等於graph[i][j]
int last[N][N];             // 記錄目前要把第i個點加入正確聯盟所需要的距離
void init(){                // 初始化
    int i, j;
    for ( i = 0 ; i < N ; i++ ){
        for ( j = i ; j < N ; j++ ){
            // 如果(i, j)有邊就會是該條邊，反之則會是INF； i == j ➔ 0
            dp[i][j] = dp[j][i] = graph[i][j];
            last[i][j] = last[j][i] = -1;          // -1代表沒經過任何點
        }
    }
}
void floyd_warshall(){
    int i, j, k;
    for ( k = 0 ; k < N ; k++ ){
        for ( i = 0 ; i < N ; i++ ){
            for ( j = i + 1 ; j < N ; j++ ){
                // 起點或終點是當前嘗試的點k 就跳過
                if ( i == k || j == k ) continue;
                if ( dp[i][k] + dp[k][j] < dp[i][j] ){    // 透過點k有更近就更新
                    dp[i][j] = dp[j][i] = dp[i][k] + dp[k][j];
                    last[i][j] = last[j][i] = k;
                }
            }
        }
    }
}
```

看完影片你必須要知道的事

- 單源最短路徑的問題定義
- Dijkstra的操作過程
- Dijkstra和Prim的相同與相異之處
- Dijkstra的使用條件
- Bellman-Ford的可用條件與操作過程
- Bellman-Ford之於偵測負環的做法
- 全點對最短路徑的問題定義
- Floyd-Warshall的DP狀態、最小子結構與遞迴式
- Floyd-Warshall的bottom-up DP的實作
- 以上三種演算法的回溯方法(找出路徑的方法)