前言：

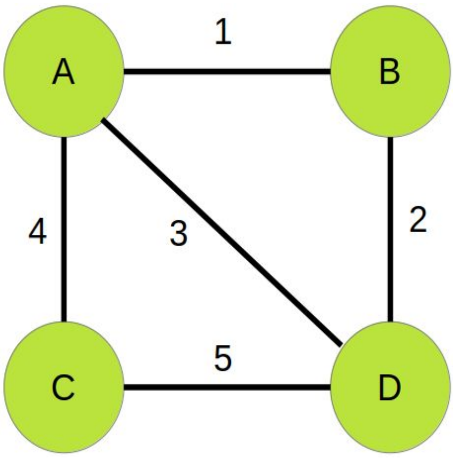
第一周的課程講述貪婪演算法 (Greedy Algorithm) 的核心理念，也就是每一步都是當下我們認為的最佳選擇，並期望這樣的方法，可以引領我們走向正確的目標。首先介紹經典的排程問題 (Scheduling Problem)，接著介紹搜尋最小生成樹 (Minimum Spanning Tree, MST) 的重要演算法：Prim's MST 演算法。作業是以 Prim's 演算法，求出 MST 的 cost。

最小生成樹 (Minimum Spanning Tree)：

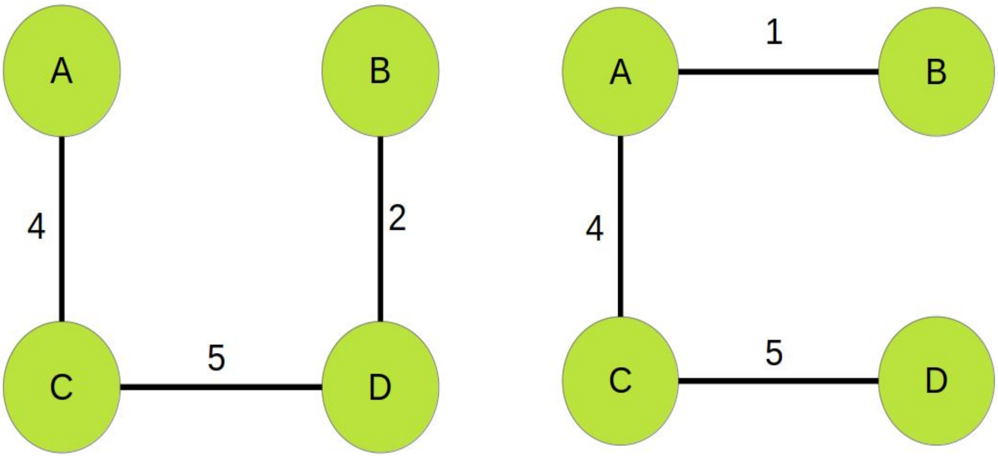
考慮一連通、帶權重的無向圖，**生成樹**的定義為：

1. 樹必須連接圖中所有節點。
2. 樹沒有迴圈
3. 因沒有迴圈且連接圖中所有點，若總共有 V 個節點，那生成數的邊總數一定為 V - 1 條。

假設有一無向圖：

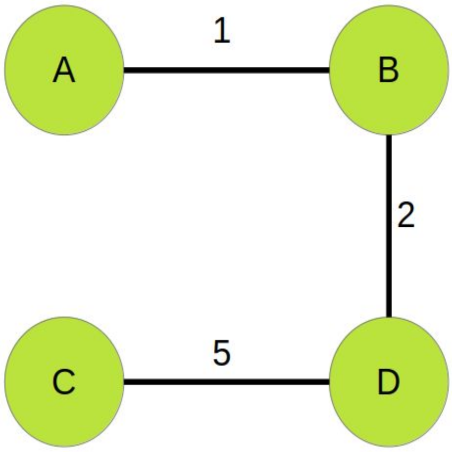


其可能的生成樹如下例。



因圖的邊有權重，所以不同的生成樹，可能有不同的權重和，其中具有最小權重和的生成樹，稱為**最小生成樹**。

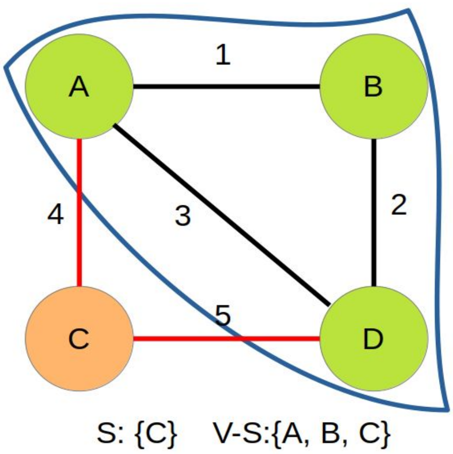
另外需注意，因最小生成樹只要求權重和最小，若圖中邊的權重並非唯一時，可能有多個最小生成樹；反之也可說，若邊的權重唯一，則最小生成樹唯一。接下來講解 Prim’s MST 演算法的概念，並證明上述性質。



Prim’s MST 演算法：

在說明前先介紹何為 cut 以及 crossing edges:

1. cut: 將圖切分為兩個非空的集合，我們稱之為 cut。如下圖藍色的線，就是一個 cut ，將圖的節點分為 S 及 V-S 兩個集合。
2. crossing edges: 若存在一條邊 (X, Y)，其中 X ⊆ S 且 Y ⊆ V − S，則我們稱該條邊為 crossing edge，所有這類邊的集合，就稱作 crossing edges。



S: {C}　V-S:{A, B, C}

Prim's 演算法的概念非常簡單，隨機選擇一個起始節點後，每次都選擇當下 cut(X, V - X) crossing edges 中權重最小的邊，加入邊集合 T 中，並將該邊位於 V - X 的節點，加入 X 中，遍歷所有節點後，在集合中的邊即形成 MST。

由上面的敘述可以發現，Prim's 演算法的基本核心，就是貪婪演算法，以下將簡述 Prim's 演算法的正確性。

Prim's 算法的證明要分成兩個部份：

1. Prim's 演算法的輸出一定是生成樹。
2. 該生成樹一定是 MST。

證明 Part I:

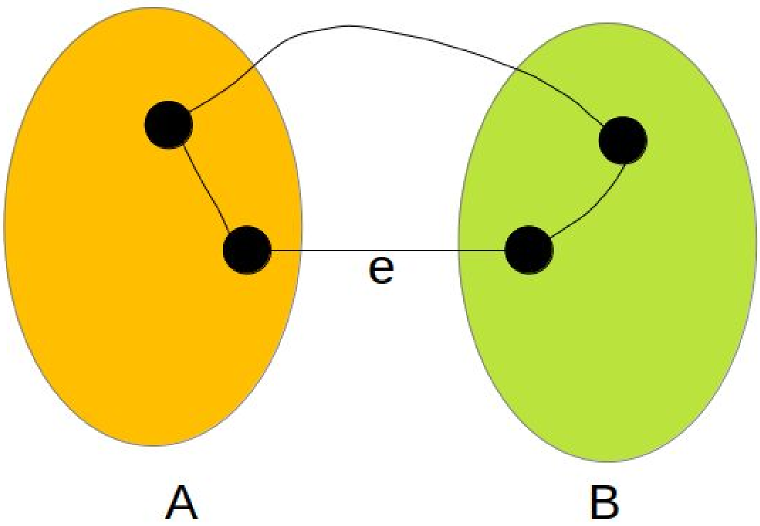
證明 Prim's 演算法的輸出是否一定是生成樹，必須先從兩個圖的基本性質說起：

1. 若且唯若圖不連通，則一定存在 cut(A, B) 沒有 crossing edges。

⇒) 假設我們從 A 及 B 中各選出一點 u 及 v，因為沒有 crossing edges，所以一定不存在路徑(u, v)，因此圖不連通。

⇐) 假設不存在 路徑(u, v)，代表說 A = { u 節點可連通的點集合}，B = { v 節點可連通的點集合}，若 cut(A, B) 有 crossing edges 的話，代表 A 與 B 集合的點之間可互相連通，與 A 及 B 的定義矛盾。

2. 若圖有迴圈 C ⊆ E，該迴圈有一條邊屬於 cut(A, B)的 crossing edge，則迴圈中一定有另一條邊亦為 cut(A, B) 的 crossing edge。
這同時說明了令一個重要的性質，若 e 為唯一跨過某個 cut(A, B) 的邊，則 e 一定不屬於任何迴圈。



有了以上兩個基本性質，Part I 的證明，就容易許多：

a. 圖 G(V, E)，若 X 為已探索過的節點集合，在 X = V 之前，算法不會停止。

⇒) 依性質 1.，若未遍歷所有節點就停止，代表有某個 cut(X, V-X) 為空集合，圖非連通。

b. 邊集合 T 一定沒有迴圈存在。

⇒) 考慮某個迭代狀態的 X 與 T 集合，加入集合 T 中的邊 e，一定是首個加入 T 的邊且跨過 cut(X, V - X)，依性質 2. ，若要形成迴圈必須要有兩條邊跨過 cut，因此 T 一定沒有迴圈存在。

綜合 a. & b. ，當遍歷所有點 V ，輸出的邊集合 T 一定是生成樹。

證明 Part II:

要證明 Prim's 演算法一定會生成 MST，關鍵在什麼狀況下，我們可以說把某條邊加入 T 集合是 **[安全的]** 。亦即證明選擇 cut(X, V - X) 中，權重最小的 crossing edge ，是否合理。

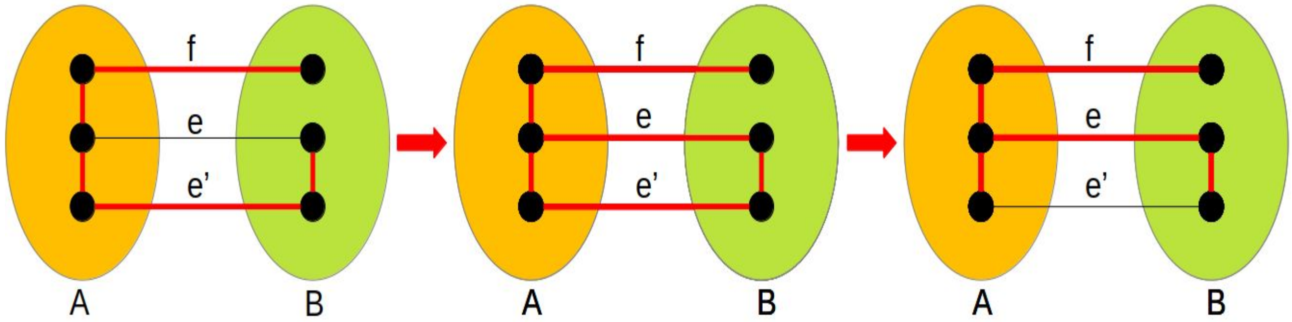
首先需證明一個性質，cut property：

考慮某個圖 G 中的一條邊 e，並假設存在一個 cut(A, B)，使得這條邊是權重最小的 crossing edge，則 e 屬於 MST。

可利用矛盾法證明，假設 e 仍是 cut(A, B) 權重最小的 crossing edge，但不屬於 MST，若我們將 e 與 MST 中的某個邊交換，且仍符合生成樹的不變量，代表新的生成樹的權重，比 MST 還要小，這與 MST 的性質矛盾，因此接下來的問題就是，該如何選擇與 e 交換的邊?交換之後仍是生成樹嗎？

a. 令 T' 為 G 的生成樹， e ∉ T' ，f ∈ T' ，則 T' ∪ { e } - { f } 仍是生成樹。

⇒) 因為 T' 必定為連通圖，若某個 cut(A, B) 有最小權重邊 e ∉ T，代表一定有另一條邊同時是 cut(A, B) 的 crossing edge 且 ∈ T'。假設將 e 加入 T' 後形成迴圈 C，依 part I 的性質 2.，生成樹中一定有另一條 cut(A, B) 的 crossing edge e' ≠ e 且 e' ∈ C，T' ∪ { e } - { e' } 仍然是生成樹，這代表我們一定可以找到一條邊 e' 與 e 交換，保持生成樹不變量，且權重和比 MST 更小 ⇒ 矛盾。



時間複雜度 (與 Dijkstra 近乎相同)：

最直接的 Prim's 演算法要遍歷所有節點(O(n))，且每個節點要遍歷所有邊(O(m))，的時間複雜度為 **O(mn)**，n 為節點數，m 為邊數，若為稠密圖 dense graph，則複雜度可能來到 O(n^3)，實在稱不上快速。

我們觀察 Prim's 演算法的步驟，每次都要找當前 cut(X, V － X)權重最小的邊，有一個資料結構非常適合這種需要不斷計算最小值的狀況，也就是 **Heap**。

＊ (Heap 的插入、刪除、取出最小值的時間複雜度皆為 **O(logn)**)

＊ (所以 Prim 與 Dijkstra 的概念近乎相同)

配合 Heap 時的算法步驟如下：

1. 各節點對應的最小權重設為無窮大，並將所有節點插入 Heap 中，時間複雜度為 O(mlogn)(注意：圖為連通，所以總邊數 m ≥ n-1)。
2. 隨機取一個節點作為起點，將該點設為已探索，更新與該節點相鄰節點的權重 (刪除及重新插入)。
3. 當尚未遍歷所有點時，則從 Heap 中取出當前最小值的點，該點設為已探索並將對應的邊加入集合 T。接著遍歷所有與該點相鄰的節點，若相鄰節點尚未探索過，則更新該點權重值 (刪除及重新插入)。
4. 遍歷所有節點後，輸出集合 T 即為 MST。

觀察上述步驟，時間複雜度主要由 Heap 相關的操作所主導：

- 預處理 **O(mlogn)**。
- 每次迭代 (共 n-1 次) 取出最小值 **O(mlogn)**。
- 每一條邊都需執行一次**刪除/重新插入 O(mlogn)**

因此搭配 Heap 的時間複雜度為 **O(mlogn)**

作業──以 Prim’s 演算法求 MST 的 cost：

問題描述：

給定一無向權重圖，每行都代表一條邊，第一與第二個欄位分別為邊的兩個節點，第三個欄位則是邊的權重，邊的權重不一定為正，且有可能值是重複的。請利用 Prim’s 演算法求出最小生成樹的權重。

註：雖然圖並不大可利用直接法 (O(mn))計算，但仍建議使用 Heap 加速，此外 Heap 需要有刪除節點的方法。

解題方法：

本次作業沒有特別需要留意的地方，以 Prim’s 演算法搭配 Heap 即可，唯一需注意的是 python 提供的套件 heapq 並不支持刪除特定節點的功能，必須自己實現。