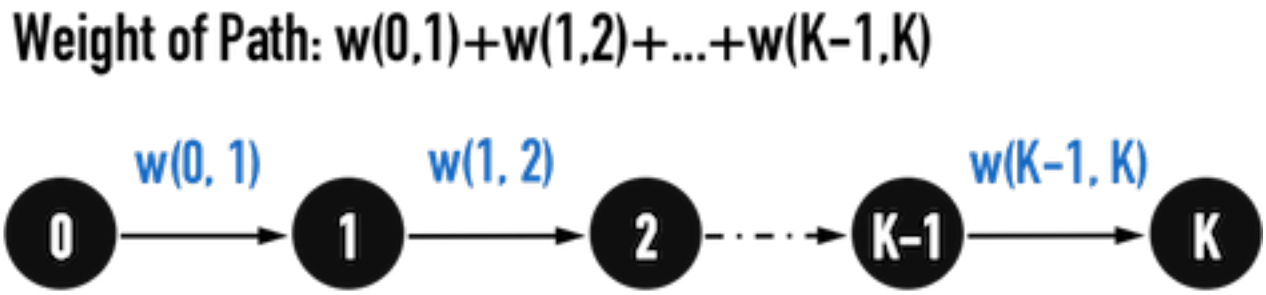
最短路徑問題(Shortest Path)

為了解決較為廣義的情境，接下來討論的最短路徑問題將考慮的是一個**weighted directed graph**，以weight總和表示path之成本，並以具有方向性之edge表示兩個vertex之間的關係。

- undirected graph的問題能夠以directed graph的模型解決，反之則無法。
- 不具weight的edge也能夠以weighted edge模擬(將全部weight設為相同數值即可)，反之則無法。
- 可以視為只能處理unweighted graph之BFS/DFS的擴充包。



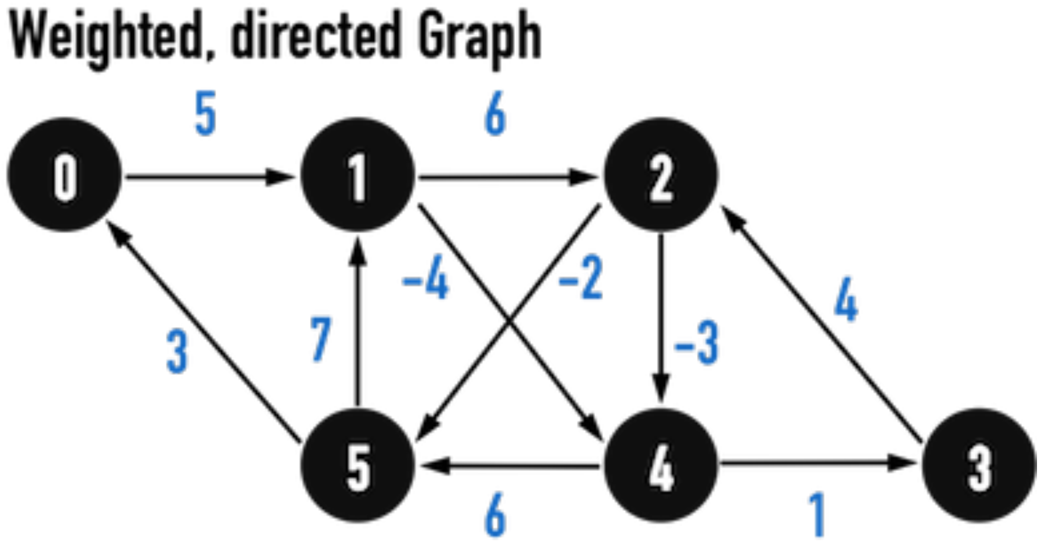
根據出發的vertex(稱為source)與終點vertex(稱為destination)之數量選擇，可以將最短路徑問題分成四類：

1. **Single-Pair Shortest Path**：從單一vertex，抵達某個特定vertex之最短路徑，此為第二種問題的子問題；
2. **Single-Source Shortest Path**：從單一vertex，抵達Graph中其餘所有vertex之最短路徑；
3. **Single-Destination Shortest Path**：從Graph中的每一個vertex抵達某個特定vertex之最短路徑：
    - 此為第二種問題之變形，只要把edge的方向相反，也就是在**GT**上，執行第二種問題之演算法即可。
4. **All-Pairs Shortest Path**：Graph中的所有vertex抵達其餘所有vertex之最短路徑。
    - 若把每一個vertex都當作起點，即可利用第二種問題之方法解決。

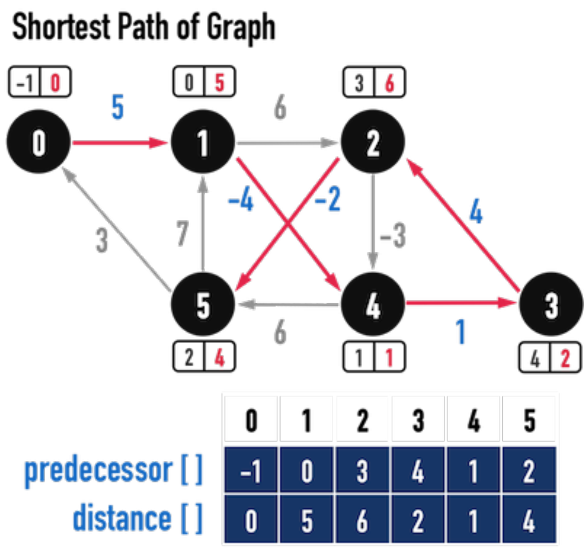
- 不過之後將介紹的**Floyd-Warshall Algorithm**有更好的答案。

綜合以上，先學第二種問題的演算法：以單一vertex為起點，抵達Graph中的其餘所有vertex之最短路徑，再學第四種問題的**Floyd-Warshall Algorithm**(以及其他高效率的演算法)，就是最明智的選擇。(推銷成功)

先來看個簡單的**Single-Source Shortest Path**範例：



- weight有負值(negative value)，所以edge越多的path，weight總和不見得越大(經過某些路徑使得整體成本降低)。



path上之**edge**越少，不見得**weight**總和越小。

在處理最短路徑問題時，最基本需要用到兩個資料項目：

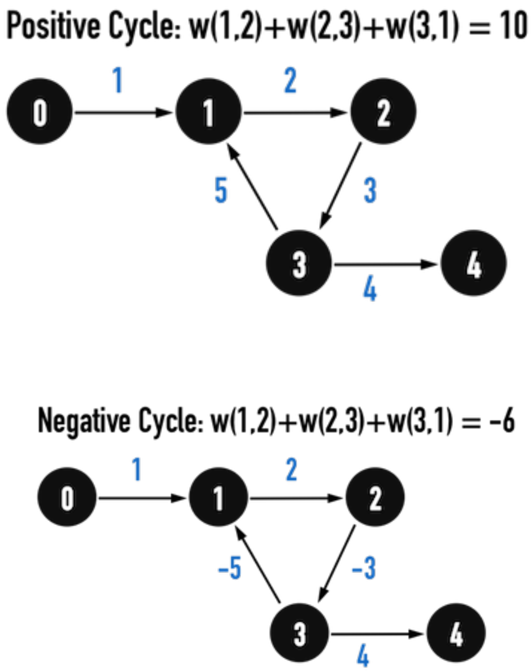
- `distance[]`：記錄從起點vertex，經過具有weight之edge，走到其餘vertex之「距離」，也就是該條path之weight。
- `predecessor[]`：除了距離之外，還希望能夠回溯路徑，因此需要記錄vertex之`predecessor`，並以此得到**Predecessor Subgraph**。

由於最短路徑一定不包含**cycle**，**Predecessor Subgraph**會是一棵**Shortest-Path Tree**。

限制

為什麼最短路徑一定不包含**cycle**？

由於weight只要求是實數(real number)，weight可正可負，因此Graph中可能出現**weight**總和為正的cycle與**weight**總和為負的cycle。，若path經過此cycle，weight之總和必定會增加，此path不會是最短路徑。



圖三(b)。

因此，在考慮最短路徑問題時，問題之Graph可以有總和為正值的cycle，但是不能有總和為負值的cycle。

而演算法所挑選出來的最短路徑之Predecessor Subgraph，一定不包含cycle。