【第十二講】

講師: 李根逸 (Ken-Yi Lee), E-mail: feis.tw@gmail.com

課程大綱

- **C++ STL** 的容器分類
- 圖
- 圖的應用
- 描述圖的結構
- 尋找路徑
  - 深度優先演算法 (DFS)
  - 廣度優先演算法 (BFS)
- 排列組合、圖與路徑搜尋樹
- 權重圖
- 最短路徑問題

C++ STL 容器分類

- 序列容器 (sequence container)：
  - 使用陣列實作：
    - **std::array, std::vector, std::deque**
  - 使用串列實作：
    - **std::forward_list, std::list**
- 容器適配器 (container adaptor)：
  - 使用其他 (序列) 容器實作：
    - **std::stack, std::queue, std::priority_queue**
- 關聯容器 (associative container)：
  - 有序型關聯容器 (使用二元搜尋樹實作)
    - **std::set, std::map, std::multiset, std::multimap**
  - 無序型關聯容器 (使用雜湊實作)
    - **std::unordered_set, std::unordered_map, std::unordered_multiset, std::unordered_multimap**

不同容器所支援的功能比較：
http://www.cplusplus.com/reference/stl/

除了容器適配器外都支援使用迭代器！

map 的使用

- **map** 是一個映射容器，將一個鍵值 **(key)**對應到一個內容值 **(value)**，所以實際上每個元素會存放兩個資料： **(key, value)**。
  - 我們怎麼表示每個元素？
    - 使用 pair 模版：

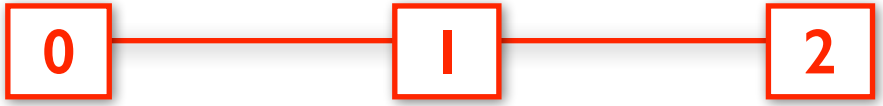
```
std::pair<key, value> e;
// e.first 表示 key 的值;
// e.second 表示 value 的值;
```

```
void ShowValues(const map<string, int> &m) {
  for (map<string, int>::const_iterator p = m.begin();
       p != m.end(); ++p) {
    cout << p->second << " ";
  }
  cout << endl;
}
```

圖

- 圖通常可以表示更廣義的結構
  - 線性容器和樹都算是圖的一種
  - 圖有『頂點』和連接頂點的『邊』構成
    - 一般邊可以分成有向邊和無向邊兩種
    - 有向邊 (單行道)

    - 無向邊 (雙向道)



      - 相當於：

圖的應用

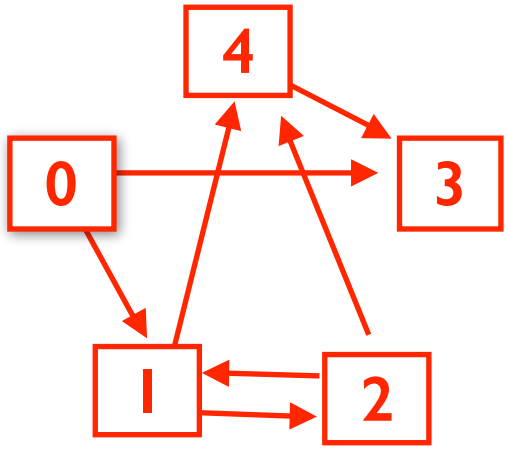
- 最短路徑問題
- 最低成本問題
- 搜尋問題

這樣的資料結構跟陣列、串列與樹有什麼不同？

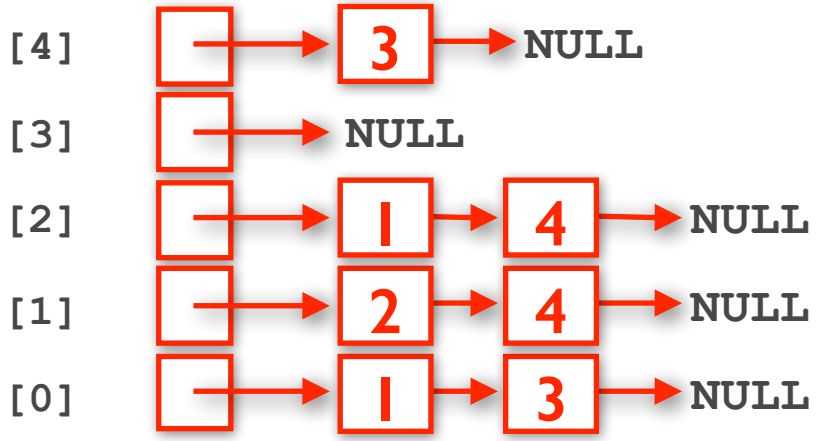
描述圖的結構 [1]

- 因為頂點與頂點之間的關係可能很複雜，一般我們可以用下面兩種方式來表示圖的結構：
  - 相鄰矩陣：
    - 用在稠密圖

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

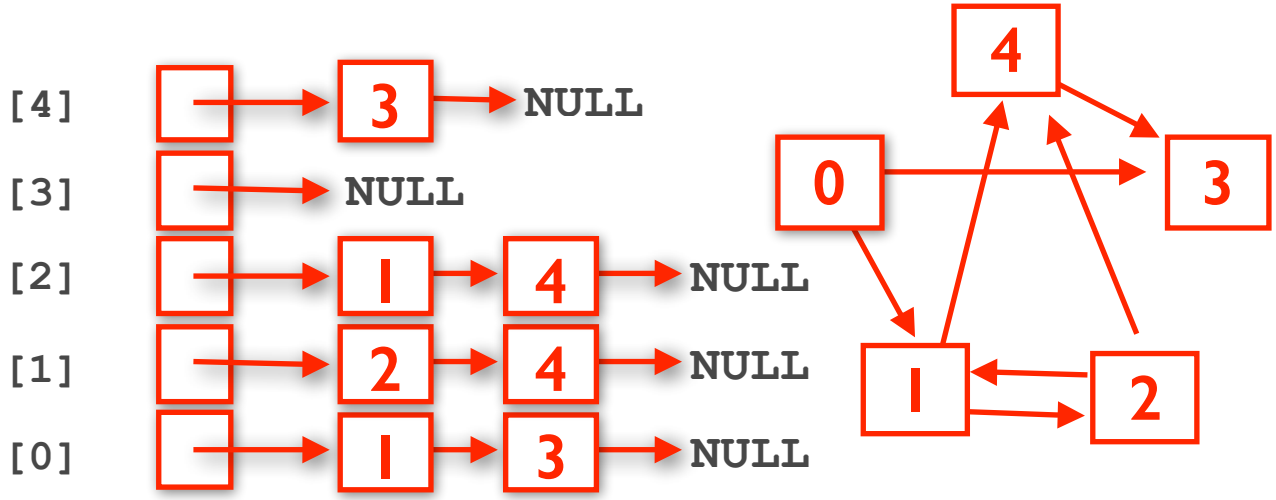


  - 相鄰頂點串列：
    - 用在稀疏圖



描述圖的結構 [2]

- 相鄰頂點串列是使用動態配置的方式，我們也可以使用多個陣列來做到相似的事情：
  - 相鄰頂點串列：



起點頂點編號：	u	0	2	1	2	1	4	0
終點頂點編號：	v	1	4	4	1	2	3	3
串列開頭邊編號：	first	6	4	1	-1	5		
下一個的邊編號：	next	-1	-1	-1	1	2	-1	0

0 1
2 4
1 4
2 1
1 2
4 3
0 3

描述圖的結構 [3]

- 在不考量效率或考慮彈性的情況下，我們可以使用下面這樣的結構來儲存頂點的相鄰關係：

```
std::map<ElemType, std::set<ElemType> > adj_;
```

  - 在這裡我們使用一個映射（**map**）將頂點對應到該頂點的鄰居集合 (**set**)
    - 我們之前提的相鄰矩陣比較像是：

```
std::vector<std::vector<ElemType> > adj_;
```

    - 我們之前提的相鄰頂點串列比較像是：

```
std::vector<std::list<ElemType> > adj_;
```

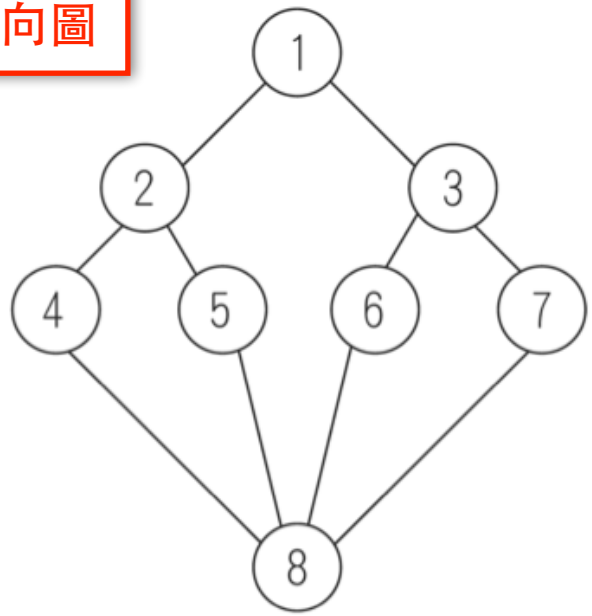
  - 使用陣列與使用映射或集合的不同？

【範例】Graph

```
typedef vector<ElemType> VectorList;
typedef map<ElemType, NeighborList> adjMap;
```

```
template<class ElemType>
class Graph {
 public:
  typedef set<ElemType> NeighborList;
  // 回傳圖內有幾個頂點
  int Size() const;
  // 將頂點 i 與頂點 j 相連
  void AddEdge(const ElemType &i,
               const ElemType &j);
  // 回傳頂點 i 與 j 是否相連
  bool IsAdjacent(const ElemType &i,
                  const ElemType &j); const
  // 回傳頂點清單
  vector<ElemType> Vertices() const;
  // 回傳頂點 i 相鄰的頂點們
  const NeighborList &Neighbors(const ElemType &i) const;
};
```

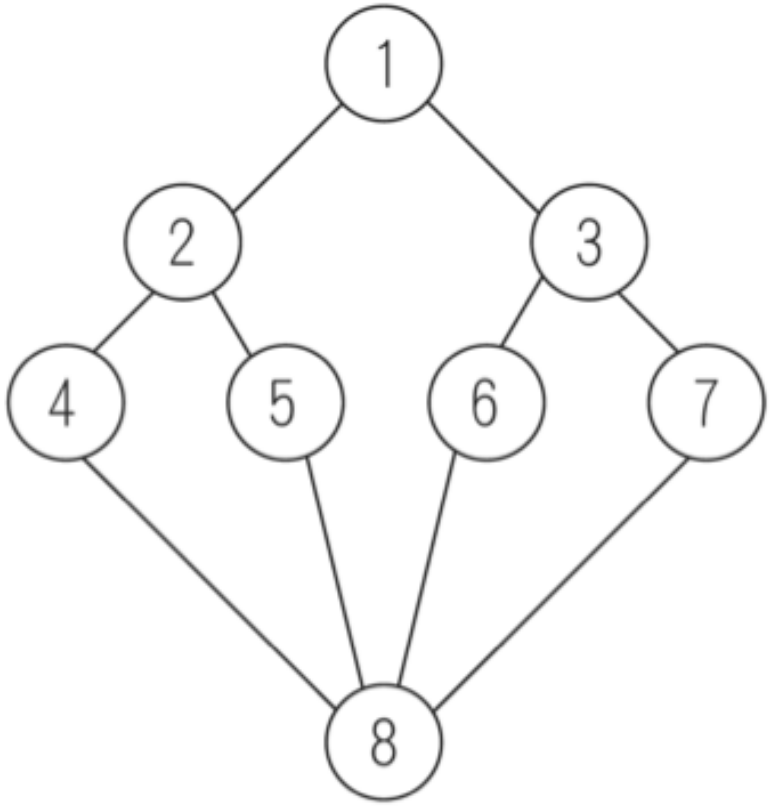
無向圖



[範例] Graph.cpp

尋找路徑

- 目標：找出由某個頂點 i 走到某個頂點 j 的可能路徑
- 常見演算法：
  - 深度優先演算法 (DFS)
    - 每次在還沒走的節點中走最深的
    - 堆疊的概念！
  - 廣度優先演算法 (BFS)
    - 每次在還沒走的節點中走最淺的
    - 佇列的概念！



深度優先演算法 (DFS)

輸入: 起點 i, 終點 j
輸出: 由 i 走到 j 的路徑

堆疊 s 表示目前可以走而還沒走的節點
陣列 path 表示目前走過的路徑
集合 explored 表示所有目前路徑走到的節點

```
s.push(i)
explored.insert(i);
當 (!s.empty()) {
  t = s.top();
  s.pop();
  path.push_back(t);
  如果 t == j:
    回傳 path
  若有不在 explored 裡且與 t 相鄰的頂點 p:
    s.push(p)
    explored.insert(p)
```

[範例] traversal.cpp

廣度優先演算法 (BFS)

輸入: 起點 i, 終點 j
輸出: 由 i 走到 j 的路徑

佇列 s 表示目前可以走而還沒走的節點
陣列 path 表示目前走過的路徑
集合 explored 表示所有目前路徑走到的節點

```
s.push(i)
explored.insert(i);
當 (!s.empty()) {
  t = s.front();
  s.pop();
  path.push_back(t);
  如果 t == j:
    回傳 path
  若有不在 explored 裡且與 t 相鄰的頂點 p:
    s.push(p)
    explored.insert(p)
```

[範例] traversal.cpp

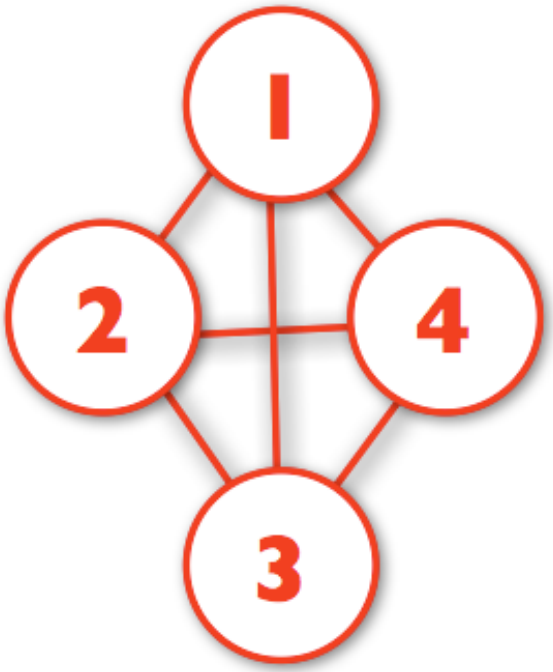
【範例】排列組合

- 試寫一程式，印出 **1** 到 **4** 四個數字所有長度的排列組合：

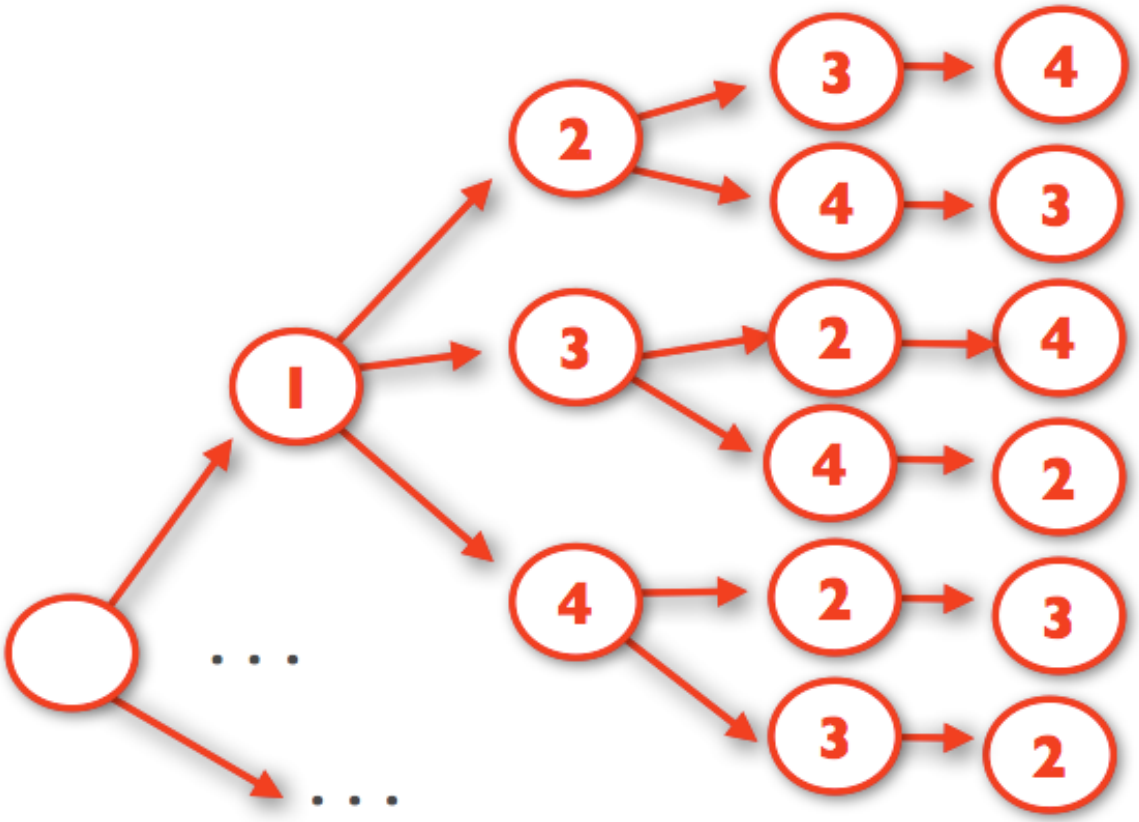
```
1
1 2
1 2 3
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 2 3
1 4 3 2
2
2 1
2 1 3
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 1 3
2 4 3 1
3
3 1
3 1 2
3 1 2 4
3 1 4 2
...
```

[範例] all_path.cpp

排列組合、圖與路徑搜尋樹



完整圖
(任兩個頂點間都有邊連結)



不重複尋訪相同頂點的路徑搜尋樹

【練習】排列組合

- 試寫一程式，印出 **1** 到 **4** 四個數字所有的排列組合：

```
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 2 3
1 4 3 2
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 1 3
2 4 3 1
3 1 2 4
3 1 4 2
3 2 1 4
3 2 4 1
3 4 1 2
3 4 2 1
4 1 2 3
4 1 3 2
...
```

[練習] ex12A.cpp

【練習】特定起點與終點

- 試寫一程式，輸入 **1** 到 **4** 之間起點的號碼與終點的號碼，然後顯示所有由起點開始到終點結束的排列組合：

範例輸入

```
1 4
```

```
1 3
```

範例輸出

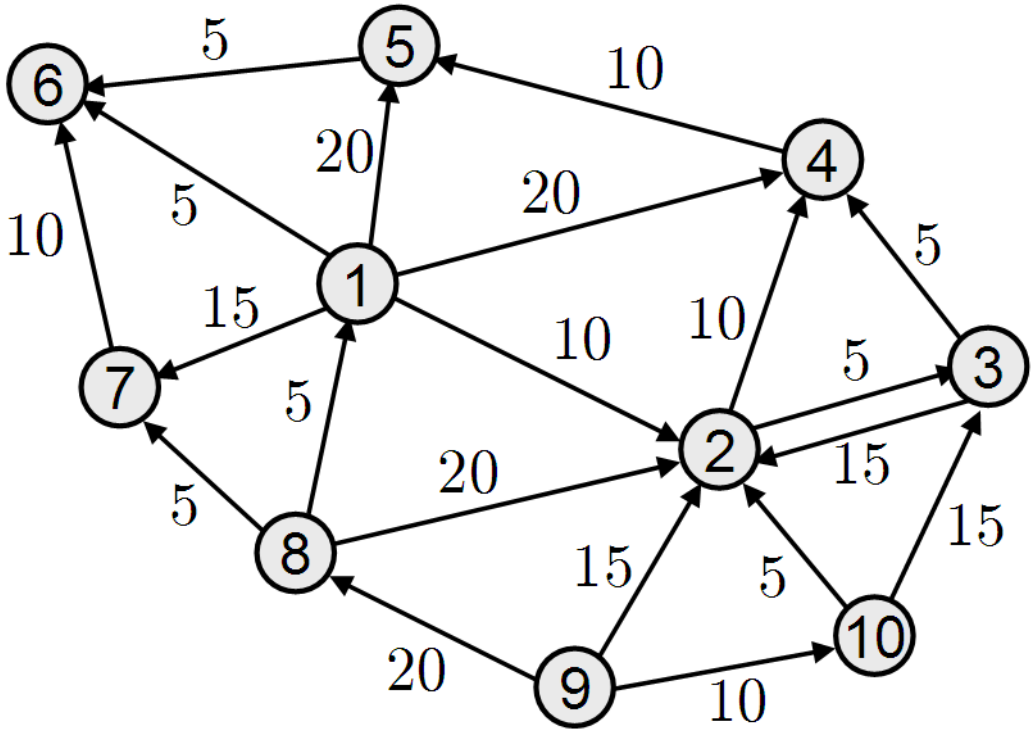
```
1 2 3 4
1 2 4
1 3 2 4
1 3 4
1 4
```

```
1 2 3
1 2 4 3
1 3
1 4 2 3
1 4 3
```

[練習] ex12B.cpp

權重圖

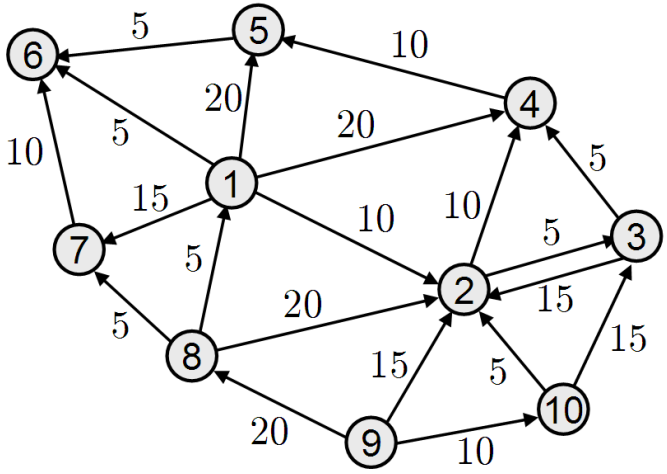
- 權重圖指的是在一般圖的邊上或頂點上給予權重值
  - 權重值可能有正有負
  - 我們要如何儲存這些權重？
    - 用無限大或 0 的權重表示未連接
    - 那如何表示無限大？
    - 用很大很大的值？



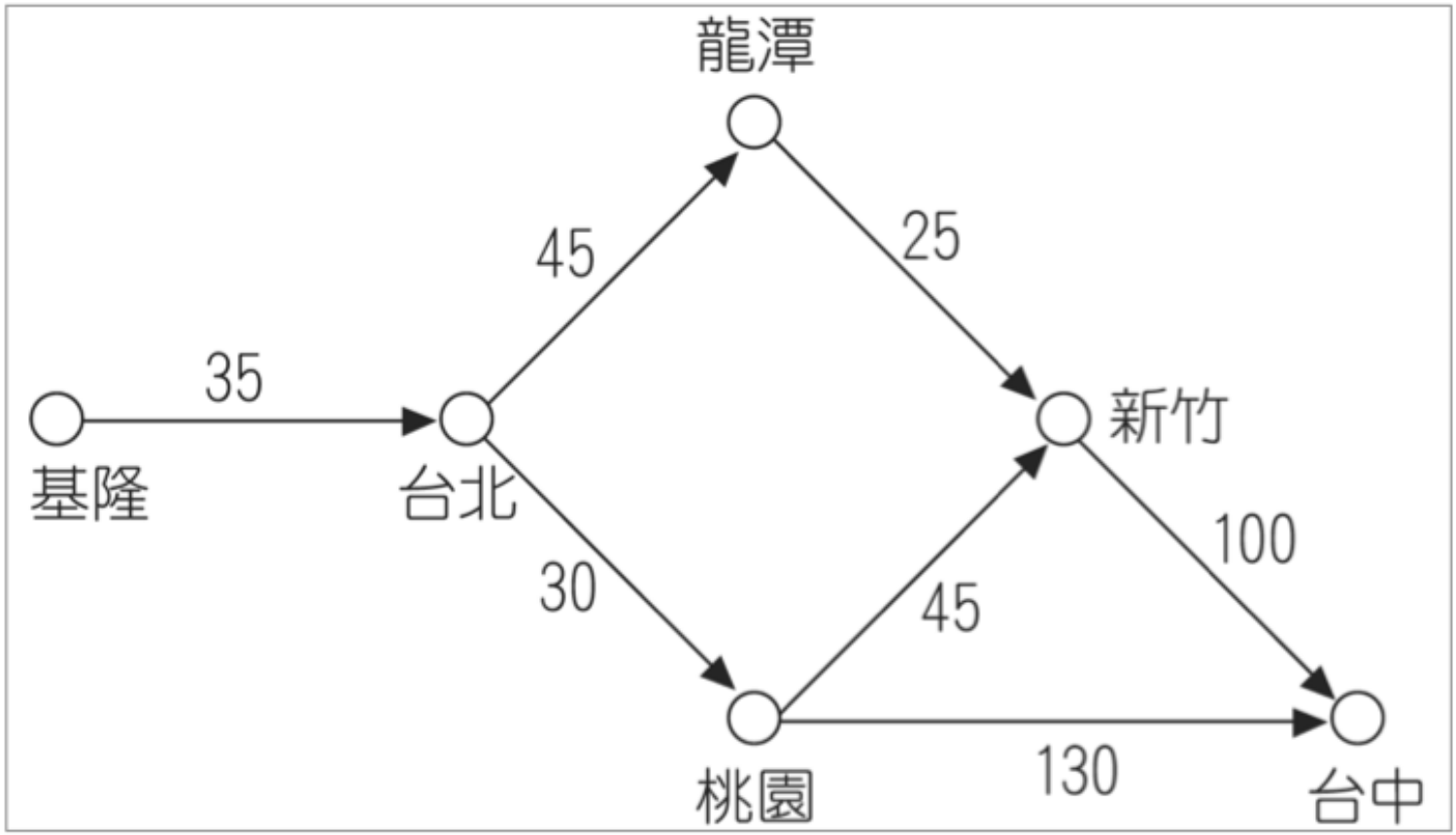
【範例】WeightedDirectedGraph

```
template<class ElemType, class WeightType>
class WeightedDirectedGraph {
 public:
  typedef set<ElemType> NeighborList;
  // 回傳圖內有幾個頂點
  int Size() const;
  // 將頂點 i 與頂點 j 相連並給權重 w
  void AddEdge(const ElemType &i,
               const ElemType &j
               WeightType w);
  // 回傳頂點 i 與 j 的權重
  WeightType GetWeight(const ElemType &i,
                       const ElemType &j); const
  // 回傳頂點清單
  vector<ElemType> Vertices() const;
  // 回傳頂點 i 相鄰的頂點們
  NeighborList Neighbors(const ElemType &i) const;
};
```

有向圖



最短路徑問題



[範例] shortest.cpp

有沒有更好的方法？

- 我們剛剛提出的是個把所有可能路徑找出來後，求出最短的。但是所有可能的路徑可能會很多很多！
- 最短路徑問題
  - 給定特定的起點跟終點時：Dijkstra 演算法
  - 求任意兩個起點跟終點的最短路徑時：Floyd-Warshall 演算法
  - 在權重可以為負的情況下，可能會有負環的出現時：Bellman-Ford 演算法和 SPFA 演算法