C/C++ 進階班
資料結構

集合
(Set)

李耕銘

課程大綱

- 集合簡介
- 集合實作方式
- 特化的集合：併查集
- 實作集合
- C++ STL 裡的集合
  - set
  - unordered_set
  - multiset

集合簡介

集合簡介

- 集合(Set)常被用在判別資料點是否在同一類
  - 資料分類/分群
  - 圖論中的連通性等具備類似性質的問題中

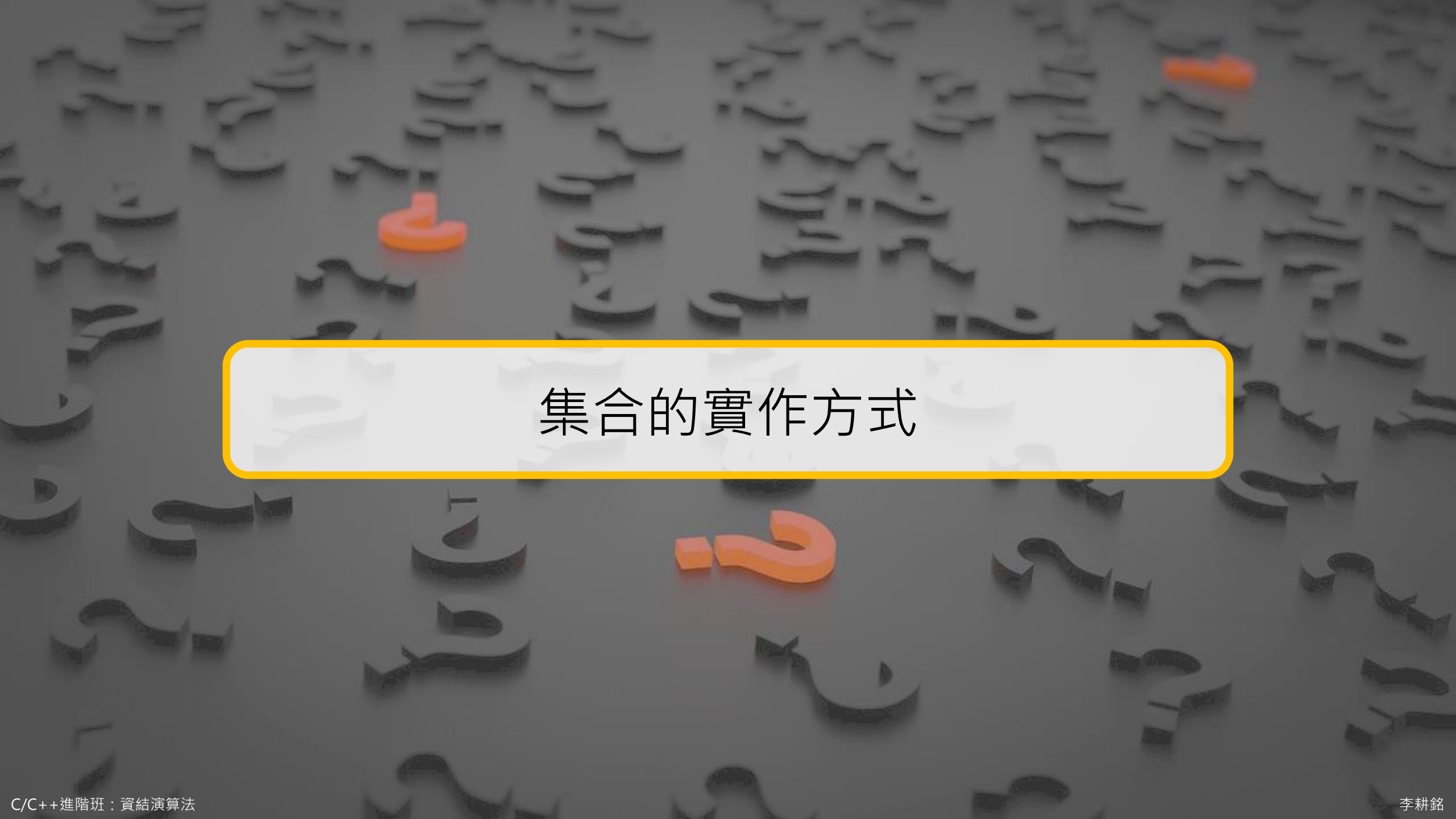
- Example
  - Set A = {1,2,3}，喜歡爬山的有 1、2、3
  - Set B = {2,4} ，喜歡逛街的有 2、4
  - Set C = {1,3,4}，喜歡讀書的有 1、3、4

集合簡介

- 集合在程式中是一種抽象資料型別（Abstract data type）
- 概念衍生自數學的「集合論」
  1. 無序性：集合內每個元素相同，之間可以沒有次序或長幼關係
     - 但實作集合時仍可定義次序關係，讓元素間按照次序排列
  2. 互異性：集合內任兩元素都不相同，每個元素只能出現一次
     - 如果需要處理同一元素出現多次的情形，使用多重集合
  3. 確定性：某元素是否屬於特定集合只有是或不是兩種答案
     - 不能有模稜兩可的情況出現

集合簡介

- 集合在數學上用來描述「等價關係」的描述
  1. 反身性(Reflexive)：
     - A「等價於」A。
  2. 對稱性(Symmetric)：
     - 若 A「等價於」B，則 B 必「等價於」A。
  3. 遞移性(Transitive)：
     - 若 A「等價於」B，且 B「等價於」C，則 A「等價於」C。
     - A, B, C 在同一集合中



集合的實作方式

集合的實作方式

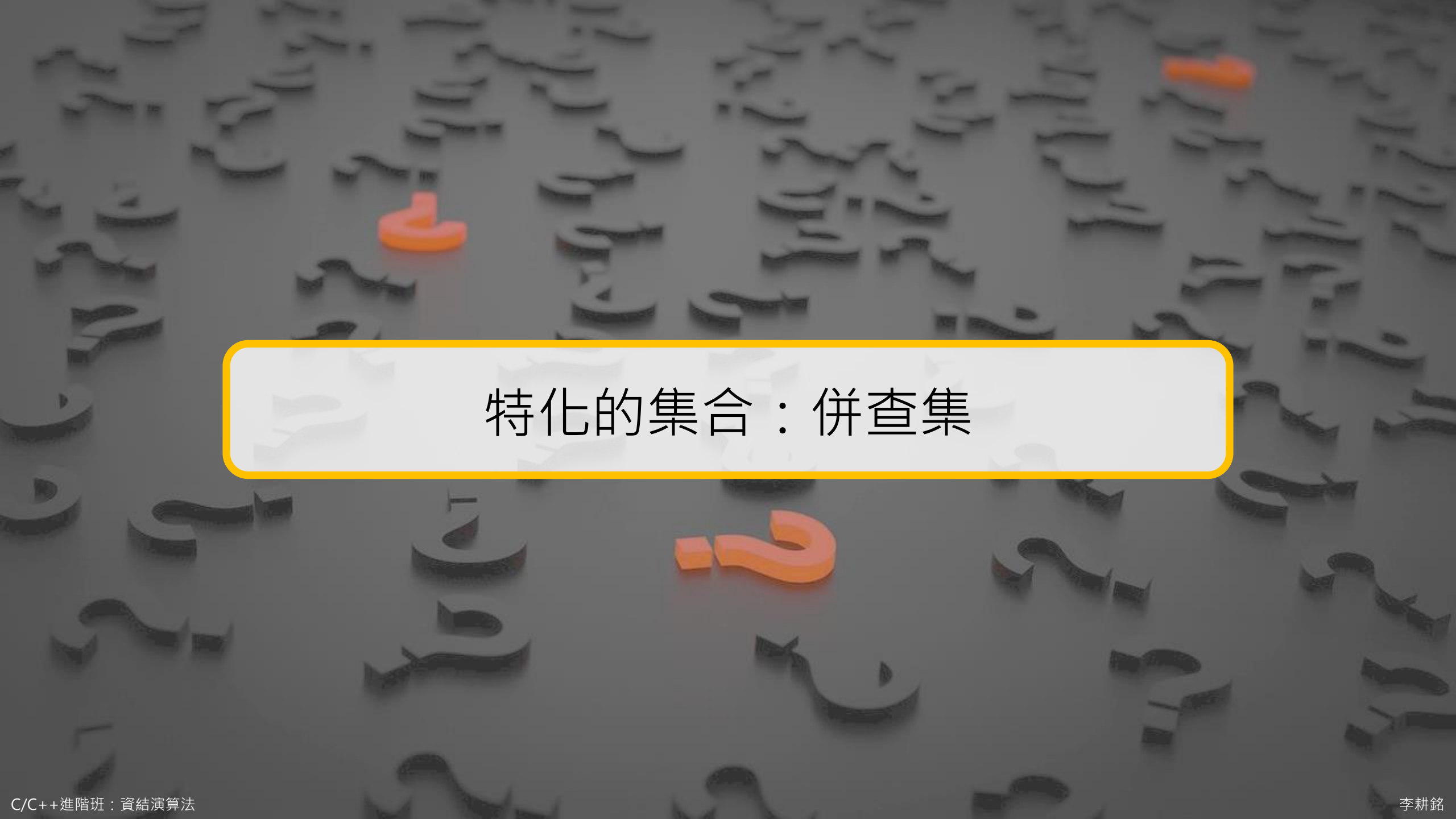
- 集合的實作方式
  - 連續的記憶體空間 (Array、Vector)
  - 鏈結串列 (Linked List)
  - 二元搜尋樹 (Binary Index Tree)
  - 二元平衡樹 (Balanced Binary Search Tree)
- 目的都是為了記錄資料所在的分類
- 不同的實作方式在進行不同操作時會有不同的複雜度

集合的實作方式

- 集合的常見操作
  1. (insert) 將元素加入集合
  2. (erase) 將元素從特定集合中刪除
  3. (count) 查詢元素是否存在於特定集合中
  4. (size) 查詢集合的大小
  5. (union) 將集合 B 合併至集合 A 中（也就是讓 B 變為空集合，而 A 則變為兩個集合的聯集）
  6. (difference) 取得兩集合的差集
  7. (intersection) 取得兩集合的交集

集合的實作方式

- 不太可能存在單一演算法可以快速實作所有的功能
- 總有部分操作會需要線性時間 (O(n)，n 為資料長度)
  - 總會犧牲部分功能作為效能上的妥協
- Example：使用向量 (vector) 儲存集合
  - 新增、查詢大小：O(1)
  - 查詢是否存在、合併、刪除：O(n)
- Example：使用二元平衡樹儲存集合
  - 新增、刪除：O($log_2 n$)
  - 查詢位置：O(n)，因把每個 set 視作獨立的二元平衡樹



特化的集合：併查集

併查集

- 在許多場合裏，集合必須滿足兩個性質：
  1. 所有不同集合間的元素都相異
     - 任兩集合的交集為空集合
  2. 在相同集合中的元素，非必要不去更動他們的位置
- 這種集合稱為「併查集 (Disjoint Sets)」
  - 每個集合都有一個代表，一筆資料也只屬於一個集合
  - 改變集合的實作方式：
    1. 原本紀錄：每個集合的內有哪些元素
    2. 改為紀錄：每個元素位在哪個集合中
- 操作複雜度 (陣列)
  - 新增、查詢、刪除：O(1)

元素	1	2	3	4	5	6	7	8	9
所在集合	A	B	C	A	A	B	C	C	B

併查集

- 若在實作併查集時，把每個元素初始化成獨立集合
  - ➢ 新增資料時將「獨立集合」與「欲加入集合」合併
- Example： 把 5 新增入 1 的集合中

  Set[5] = Set[1]
  - ➢ 合併僅須：O(1)？

元素	1	2	3	4	5	6	7	8	9
所在集合	A	A	A	D	E	F	G	H	I

元素	1	2	3	4	5	6	7	8	9
所在集合	A	A	A	D	A	F	G	H	I

併查集

- 若在實作併查集時，把每個元素初始化成獨立集合
  - 新增資料時將「獨立集合」與「欲加入集合」合併
- Example： 把 1 新增入 5 的集合中
  - 合併其實須要：O(n)！
  - 因為要改變所有跟 1 屬於同集合的資料！

元素	1	2	3	4	5	6	7	8	9
所在集合	A	A	A	D	E	F	G	H	I

元素	1	2	3	4	5	6	7	8	9
所在集合	E	E	E	D	E	F	G	H	I

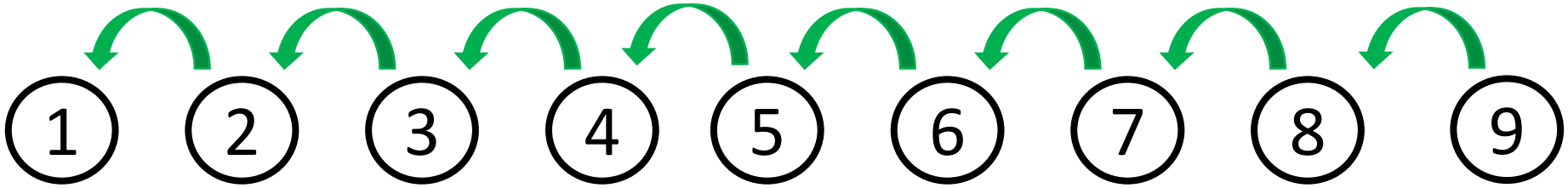
併查集

- 改記錄「與誰合併」的資訊
  - ➢ 可以把合併的複雜度降成 O(1)
- Example：
  1. 把 5 加入 1
  2. 把 6 加入 5
  3. 把 7 加入 6
  4. 把 8 加入 7
- 合併的複雜度成功降成 O(1) 了！

併查集

- 合併的複雜度降成 O(1)，但查詢變成 O(n), err.....
- Example：
  - 查詢 2~9 的所屬集合需要遞迴很多次 :(

元素	1	2	3	4	5	6	7	8	9
集合	1	1	2	3	4	5	6	7	8



併查集

- 解決方案
  - 把經過的資料集合都直接指到底
  - 可以把之後查詢的複雜度降到 O(1)
- Example：
  1. 把 5 加入 1
  2. 把 6 加入 5，把 6 改成 1
  3. 把 7 加入 6，把 7 改成 1
  4. 把 8 加入 7，把 8 改成 1

元素	1	2	3	4	5	6	7	8	9
集合	1	2	3	4	5	6	7	8	9

元素	1	2	3	4	5	6	7	8	9
集合	1	2	3	4	1	6	7	8	9

元素	1	2	3	4	5	6	7	8	9
集合	1	2	3	4	1	1	7	8	9

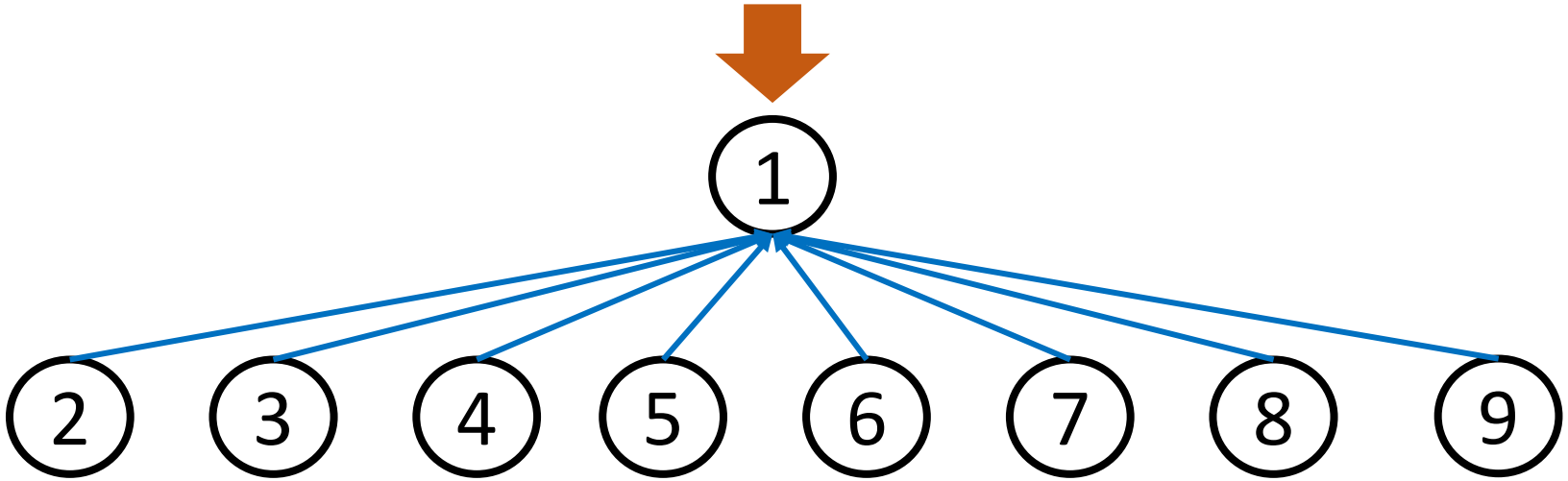
元素	1	2	3	4	5	6	7	8	9
集合	1	2	3	4	1	1	1	8	9

元素	1	2	3	4	5	6	7	8	9
集合	1	2	3	4	1	1	1	1	9

併查集

- 解決方案：把沿途經過的資料集合都直接指到底
  - 變成一顆樹
  - 可用根節點來表示目前所屬哪個集合

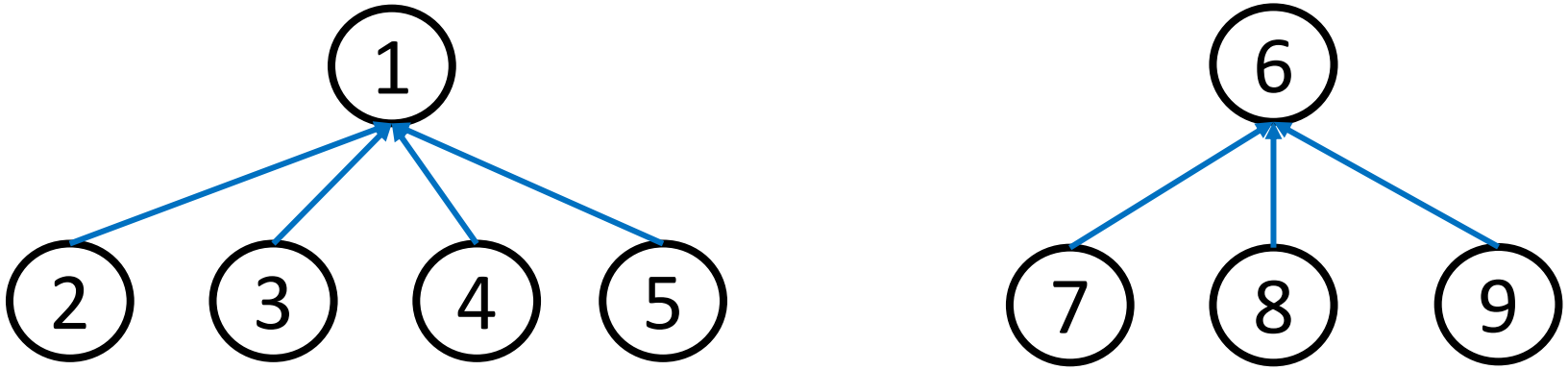
元素	1	2	3	4	5	6	7	8	9
集合	1	1	1	1	1	1	1	1	1



併查集

- 如何合併兩集合呢？
  - 改變該集合根節點的所屬集合
  - Example：把 6 加到 1 之下

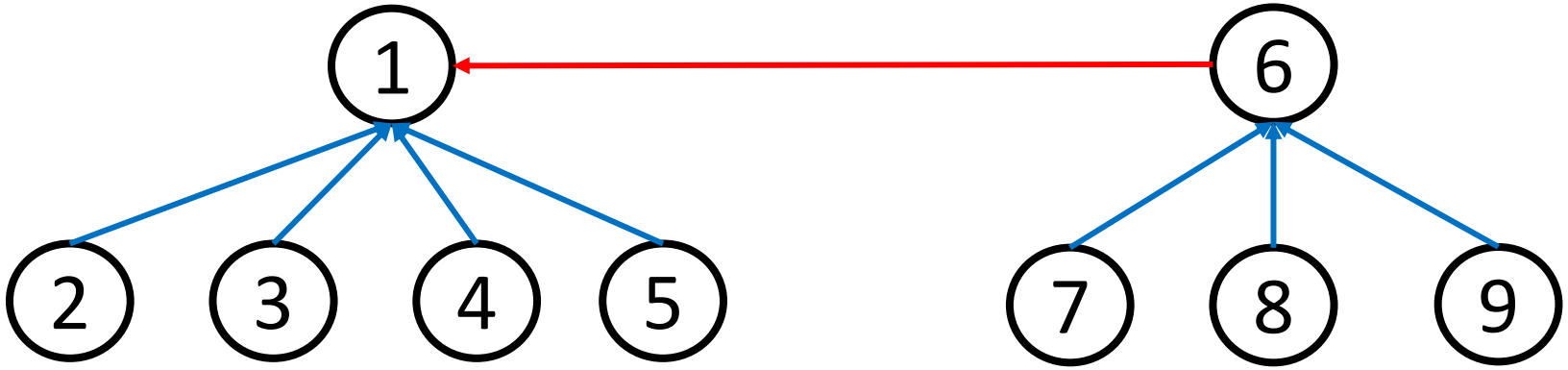
元素	1	2	3	4	5	6	7	8	9
集合	1	1	1	1	1	6	6	6	6



併查集

- 如何合併兩集合呢？
  - 改變該集合根節點的所屬集合
  - Example：把 6 加到 1 之下

元素	1	2	3	4	5	6	7	8	9
集合	1	1	1	1	1	1	6	6	6



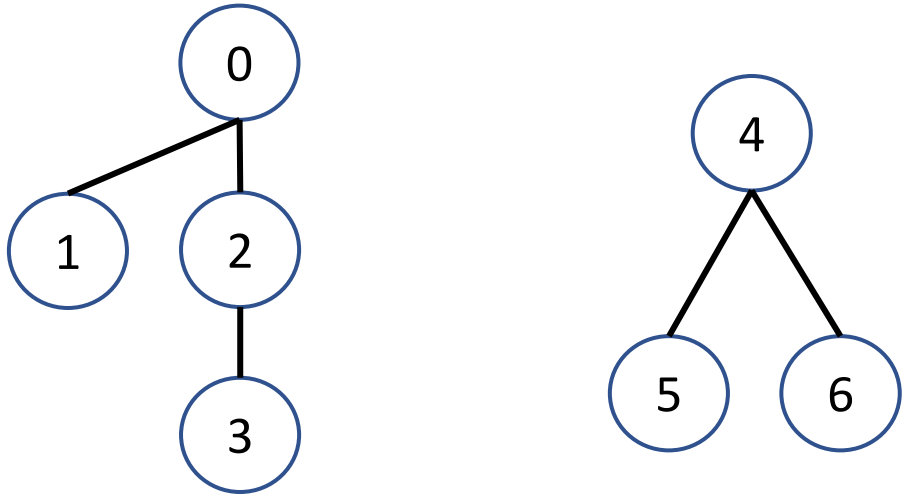
實作併查集

實作併查集

Q：如何有效率的區分集合/分組？

A：把每一組建立成一棵樹

根節點決定了屬於哪個集合



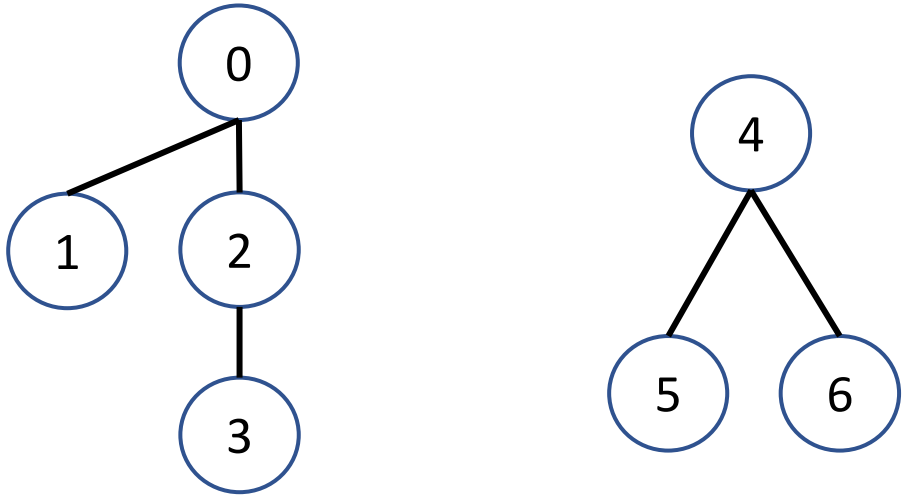
實作併查集

以陣列實作集合 set 表示

1. set[v] ≥ 0時，表示頂點 v 的 predecessor 編號
2. set[v] < 0 時，表示該集合有 |set[v]| 個頂點

Example：

Vertex	0	1	2	3	4	5	6
set	-4	0	0	2	-3	4	4



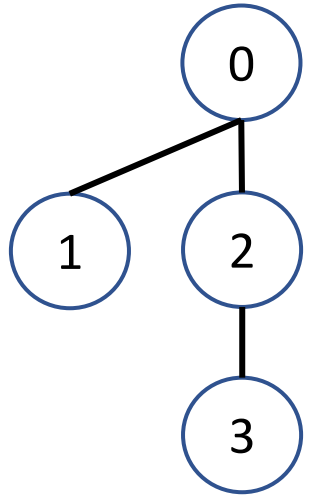
實作併查集

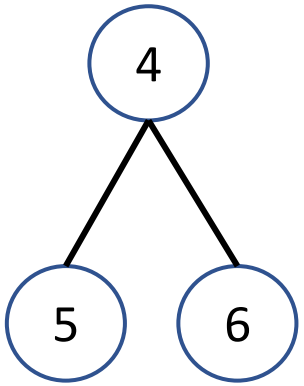
Q：如何找到 v 從屬於哪個集合？

1. 若 set[v] ≥ 0，則令 v = set[v]
2. 繼續往上找 set[v]
3. 直到 set[v] < 0

find_set

```
1  find_set(set, v){
2      while(set[v] ≥ 0)
3          v = set[v]
4      return v
5  }
```



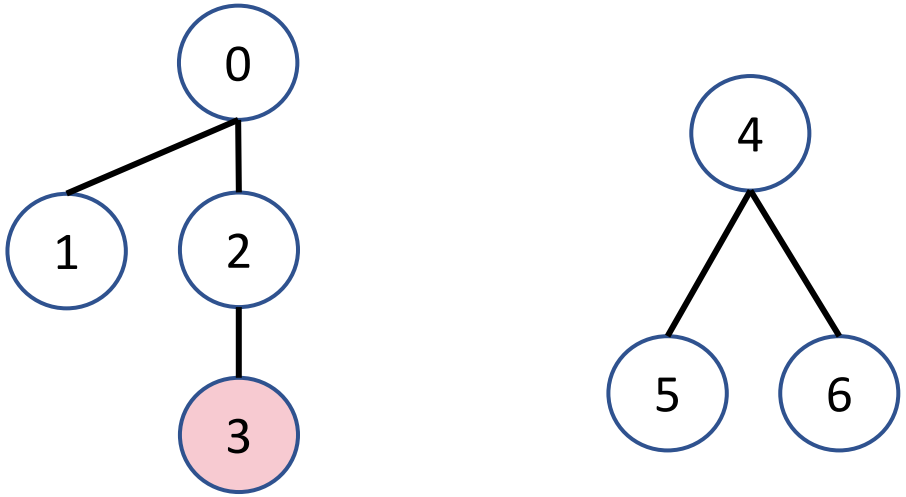


Vertex	0	1	2	3	4	5	6
set	-4	0	0	2	-3	4	4

實作併查集

Collapsing

- 為了增進效能，把**樹高塌陷成 1**
- 只要經過 1 次搜尋就能知道該點屬於哪個集合
- 往根節點 root 中經過的頂點 v 都設定成：
  - set[v] = root



find_set_collapsing

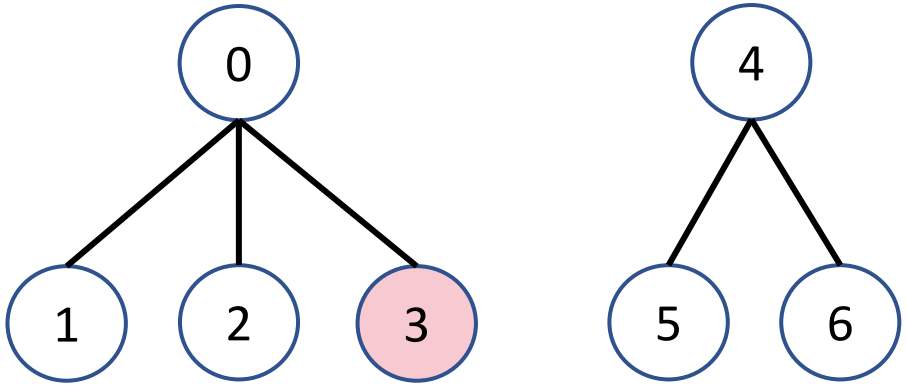
```
1  find_set_collapsing (set, v){
2      root = v
3      while(set[root] ≥ 0)
4          root = set[root]
5      while(v != root)
6          predecessor = set[v]
7          set[v] = root
8          v = predecessor
9      return root
10 }
```

Vertex	0	1	2	3	4	5	6
set	-4	0	0	2	-3	4	4

實作併查集

Collapsing

- 為了增進效能，把**樹高塌陷成 1**
- 只要經過 1 次搜尋就能知道該點屬於哪個集合
- 往根節點 root 中經過的頂點 v 都設定成：
  - set[v] = root



find_set_collapsing

```
1  find_set_collapsing (set, v){
2      root = v
3      while(set[root] ≥ 0)
4          root = set[root]
5      while(v != root)
6          predecessor = set[v]
7          set[v] = root
8          v = predecessor
9      return root
10 }
```

Vertex	0	1	2	3	4	5	6
set	-4	0	0	0	-3	4	4

實作併查集

Q：如何新增/合併兩集合？

A：把 root 的 set 指向另一 root 即可！

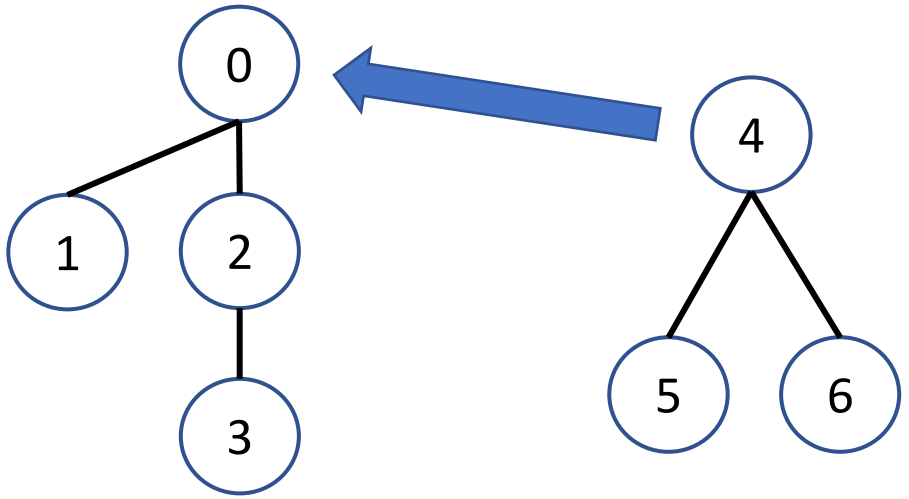
Q：如何決定誰要從屬於誰？

A：通常個數越多的 set，樹高越大

(但其實不一定 :( )

讓個數少的 set(4) 從屬於個數多的 set(0)

最後記得讓根節點更新成 set(0)+set(4)



Vertex	0	1	2	3	4	5	6
set	-4	0	0	2	-3	4	4

實作併查集

Q：如何新增/合併兩集合？

A：把 root 的 set 指向另一 root 即可！

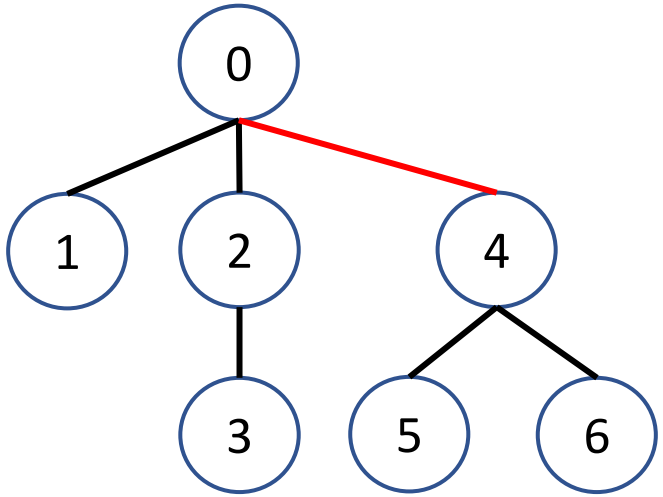
Q：如何決定誰要從屬於誰？

A：通常個數越多的 set，樹高越大

(但其實不一定 :( )

讓個數少的 set(4) 從屬於個數多的 set(0)

最後記得讓根節點更新成 set(0)+set(4)



Vertex	0	1	2	3	4	5	6
set	-7	0	0	2	0	4	4

實作併查集

Q：如何新增/合併兩集合？

A：把 root 的 set 指向另一 root 即可！

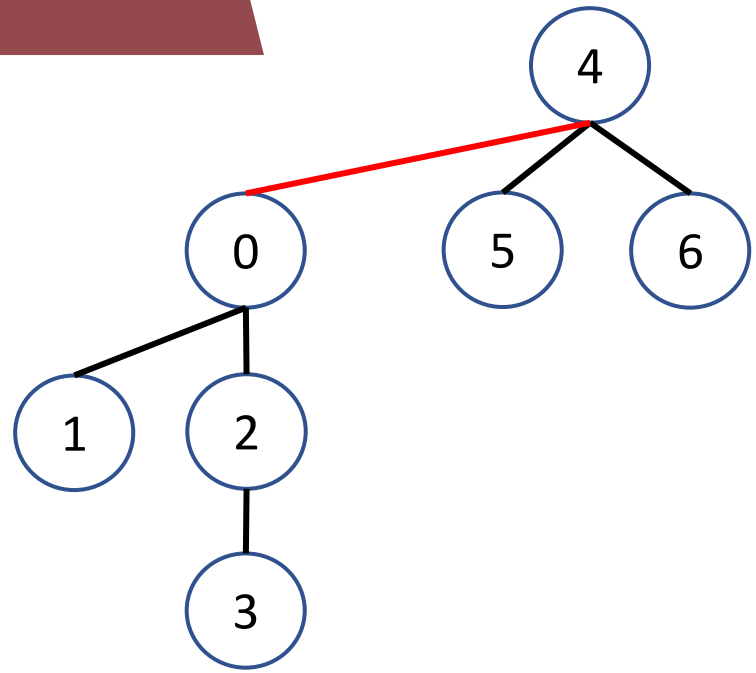
Q：如何決定誰要從屬於誰？

A：通常個數越多的 set，樹高越大

(但其實不一定 :( )

讓個數少的 set(4) 從屬於個數多的 set(0)

最後記得讓根節點更新成 set(0)+set(4)



Vertex	0	1	2	3	4	5	6
set	4	0	0	2	-7	4	4

反過來樹高通常會更高 :(

實作併查集

Q：如何新增/合併兩集合？

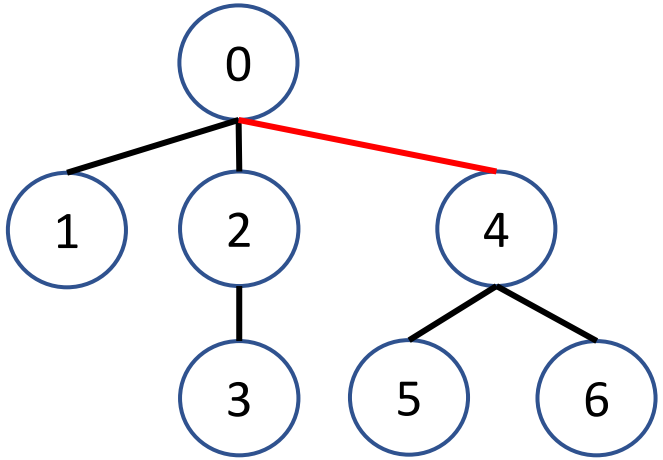
A：把 root 的 set 指向另一 root 即可！

Q：如何決定誰要從屬於誰？

A：通常個數越多的 set，樹高越大 (但不一定 :( )

讓個數少的 set(4) 從屬於個數多的 set(0)

最後記得讓 root 的 set 更新成 set(0)+set(4)



merge_set

```
1  merge_set (set, u, v){
2      u_root = find_set_collapsing(set,u)
3      v_root = find_set_collapsing(set,v)
4      if(set[u_root] ≤ set[v_root])
5          set[u_root]  += set[v_root]
6          set[v_root] = u_root
7      else
8          set[v_root]  += set[u_root]
9          set[u_root] = v_root
10 }
```

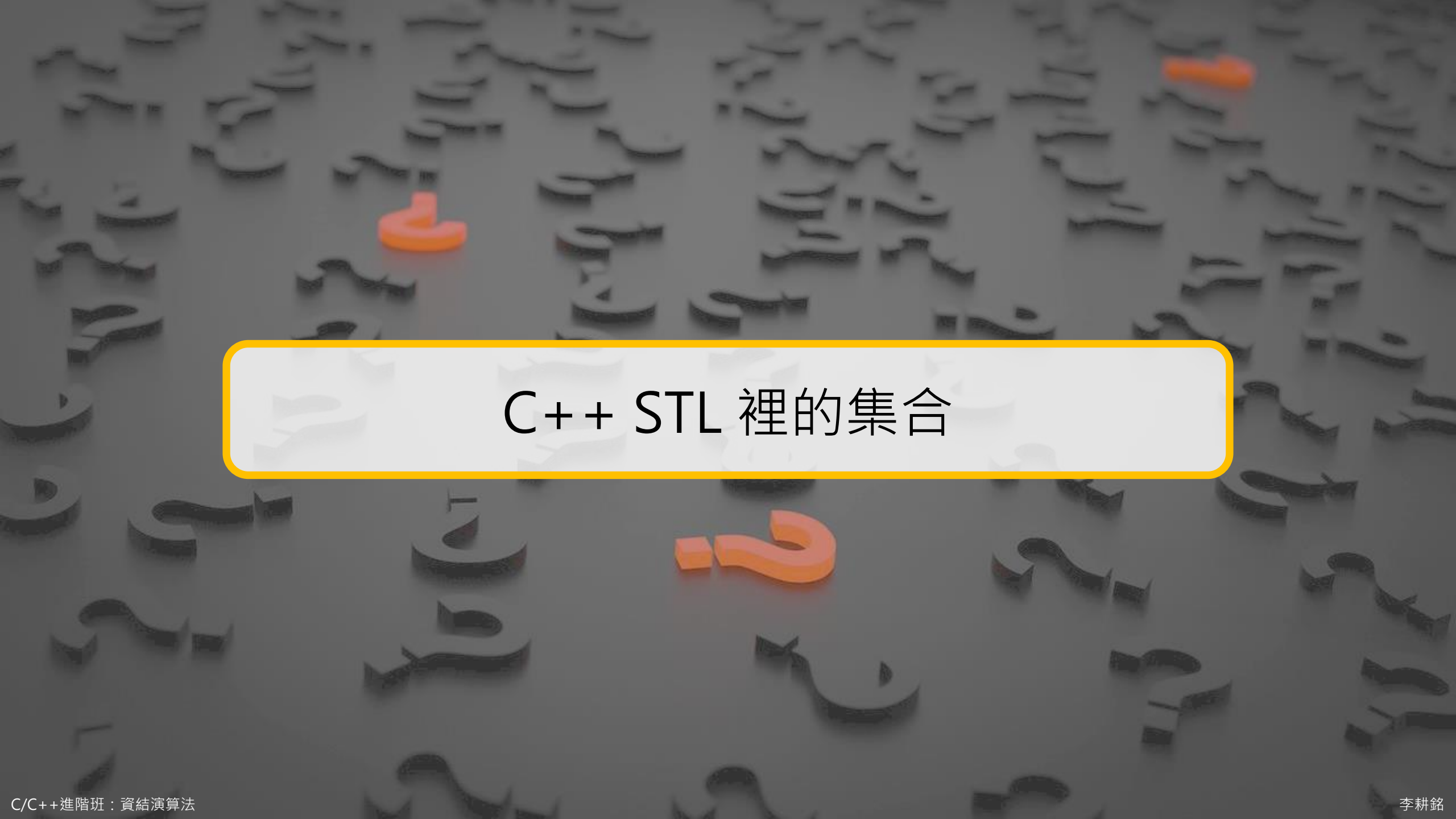
Vertex	0	1	2	3	4	5	6
set	-7	0	0	2	0	4	4

Example Code

Mission

以類別與向量 (vector) 實作集合，並完成以下函式：

1. 建構式
2. 新增資料至某資料的集合
3. 搜尋資料所在集合



C++ STL 裡的集合

Map 與 Set 的差異

Map

1. Key→Value
2. 資料結構：雜湊表或紅黑樹
3. 常用來儲存對應關係

Set

1. Value
2. 資料結構：雜湊表或紅黑樹
3. 常用來分群、紀錄出現與否

C++ STL 裡的集合

- unordered_map、 unordered_set 是 hash table
  - 沒有次序關係
  - 插入、搜尋、刪除：$O(1)$
- map、set 是紅黑樹(Red-Black Tree)
  - 有次序 (依照左右節點區分)
  - 不能重複
  - 插入、搜尋、刪除：$O(log_2 N)$
- multimap、multiset 是紅黑樹(Red-Black Tree)
  - 有次序 (依照左右節點區分)
  - 可以重複
  - 插入、搜尋、刪除：$O(log_2 N)$

set 與 unordered_set

set

- #include <set>
- 原理：紅黑樹(Red-Black Tree)
- 優：有次序
- 缺：占用多的空間
- 速度：較慢($O(log_2 N)$)
- 適用：有順序要求的資料

unordered_set

- #include <unordered_set>
- 原理：雜湊表(Hash table)
- 優：速度快
- 缺：沒有次序資料、空間需求更大
- 速度：較快($O(1)$)
- 適用：沒有次序的資料

C++ STL 裡的 set

- insert(x)
  - ✓ 把資料 x 放進集合
- erase(x)
  - ✓ 把資料 x 從集合中移除
- count(x)
  - ✓ 檢查資料 x 是否有在集合中
  - ✓ 1：有、0：無
- clear()
  - ✓ 清空集合

- 跑遍所有資料

```
set<int>::iterator it;
for(it=s.begin();it! =s.end();it++)
    cout<<*it<<endl;
```

- 找尋資料

```
if(s.find(data)! =s.end())
    cout << "Found" << endl;
else
    cout << "Not Found. " << endl;
```

C++ STL 裡的 set

Set 特殊用法：

- 宣告時可指定排序方式
- 預設是 less 或升冪
- unordered_set 無此用法！

```
set<int> my_set;

for (int i=0; i<10; i++)
    my_set.insert(i);
```

0	1	2	3	4	5	6	7	8	9

```
set<int, greater<int>> > my_set;

for (int i=0; i<10; i++)
    my_set.insert(i);
```

9	8	7	6	5	4	3	2	1	0

C++ STL 裡的 set

Set 特殊用法：

- upper_bound(x)：回傳大於x 的迭代器
- lower_bound(x) ：回傳不小於 x 的迭代器


```
4
3
```

```
#include <iostream>
#include <set>
using namespace std;

int main (){
   set<int> my_set;

   for (int i=0; i<10; i++)
      my_set.insert(i);

   cout << *(my_set.upper_bound(3)) << endl;
   cout << *(my_set.lower_bound(3)) << endl;

   return 0;
}
```

C++ STL 裡的 set

Set 特殊用法：

- upper_bound(x)：回傳大於x 的迭代器
- lower_bound(x) ：回傳不小於 x 的迭代器

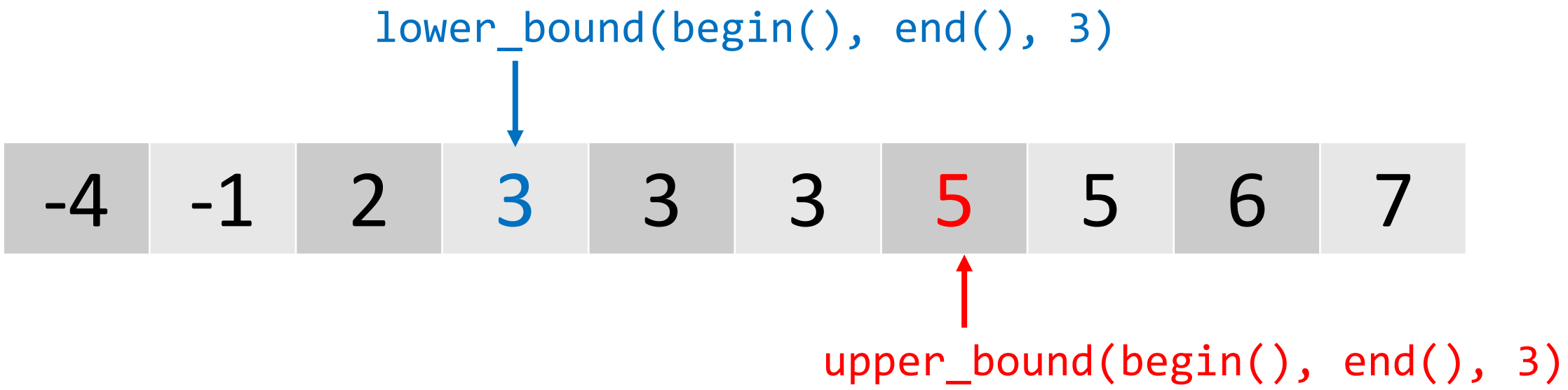

```
2
3
```

```
#include <iostream>
#include <set>
using namespace std;

int main (){
   set<int,greater<int>> my_set;

   for (int i=0; i<10; i++)
      my_set.insert(i);

   cout << *(my_set.upper_bound(3)) << endl;
   cout << *(my_set.lower_bound(3)) << endl;

   return 0;
}
```

C++ STL 裡的 set

Set 特殊用法：

- upper_bound(x)：回傳大於x 的迭代器
- lower_bound(x) ：回傳不小於 x 的迭代器



unordered map & set

- unordered map 跟 unordered set
  - 都沒有支援迭代器運算(++、--)
  - 也沒有 upper 或 lower bound ，因為沒有次序！！
  - 但可使用 ranged based for loop 取出資料

```
for (auto it: unordered_set/unordered_map) {
    // Do something
}
```

Set & Multiset

- Map/Set 不允許插入重複的資料
- Multimap/Multiset 允許插入重複的資料
  - #include <map>
  - 插入的資料會以二元平衡樹儲存(紅黑樹)
    - ✓ 可以保障有次序性
    - ✓ 每次新增、查詢、刪除：$O(log_2 n)$
  - 可以當作陣列使用，但 Multimap 不支援下標[]存取！
  - 嚴格來說 multiset 並非集合！
- Priority_Queue
  - 取出元素：O(n)

Multimap/Multiset

初始化

Multimap 不支援下標[]存取

```
multimap<int, string> my_multimap;
multimap<int, string>::iterator iter, start, finish;

my_multimap.insert(pair<int, string>(0, "David"));
my_multimap.insert(pair<int, string>(0, "Rallod"));
my_multimap.insert(pair<int, string>(1, "Mick"));
my_multimap.insert(pair<int, string>(1, "John"));
my_multimap.insert(pair<int, string>(2, "Daphene"));
my_multimap.insert(pair<int, string>(3, "Lily"));
cout << "Size of this multimap: " << my_multimap.size() << endl;
cout << endl;

cout << "Content of this multimap:" << endl;
for (iter = my_multimap.begin(); iter != my_multimap.end(); iter++)
    cout << iter->first << " " << iter->second << endl;
cout << endl;
```

Multimap/Multiset

搜尋

```
cout << "Search for index 0 in this multimap:" << endl;
int counts = my_multimap.count(0);
cout << "Number of index 0: " << counts << endl;
iter = my_multimap.find(0);
for (int i = 0; i < counts; i++, iter++)
    cout << iter->first << " " << iter->second << endl;
cout << endl;

cout << "Search for index 0~2 in this multimap:" << endl;
start = my_multimap.lower_bound(0); // >=0
finish = my_multimap.upper_bound(2); // >2
for (; start != finish; start++)
    cout << start->first << " " << start->second << endl;
cout << endl;
```

Multimap/Multiset

刪除

```
cout << "Delete for index 1 in this multimap:" << endl;
   iter = my_multimap.begin();
   while(true){
       iter = my_multimap.find(1);
       if(iter!=my_multimap.end())
           my_multimap.erase(iter);
       else
           break;
   }
   cout << "Size of this multimap: " << my_multimap.size() << endl;
   cout << my_multimap.size() << endl;
   cout << "Content of this multimap:" << endl;
   for (iter = my_multimap.begin(); iter != my_multimap.end(); iter++)
       cout << iter->first << " " << iter->second << endl;

   cout << "Clear this multimap!" << endl;
   if (!my_multimap.empty())
       my_multimap.clear();
   cout << "Size of this multimap: " << my_multimap.size() << endl;
```

Set & Multiset

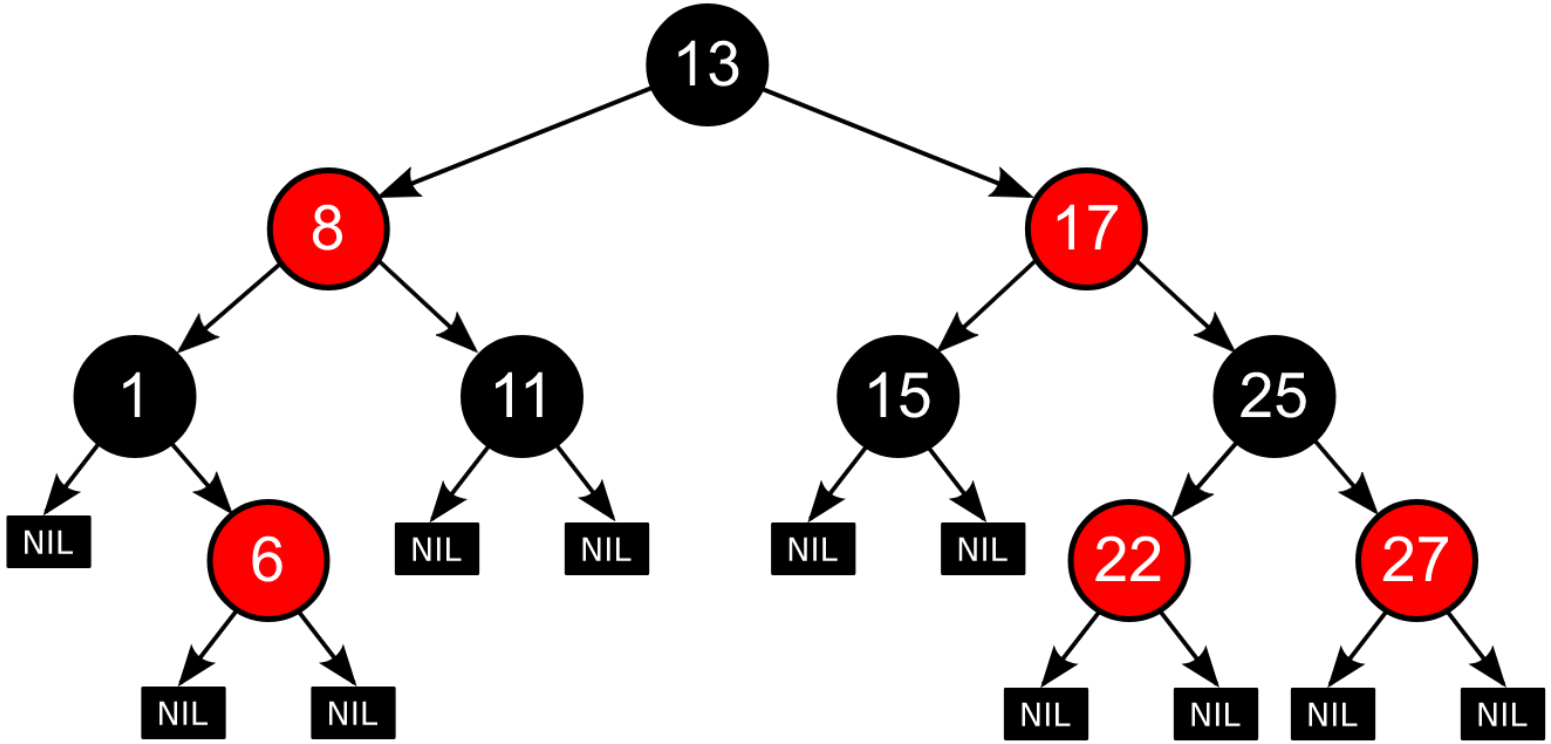
	map	set	unordered map	unordered set	multimap	multiset	unordered multimap	unordered multiset
儲存資料	key→value	value	key→value	value	key→value	value	key→value	value
資料結構	紅黑樹	紅黑樹	雜湊表	雜湊表	紅黑樹	紅黑樹	雜湊表	雜湊表
新增複雜度	$O(log_2 n)$	$O(log_2 n)$	$O(1)$	$O(1)$	$O(log_2 n)$	$O(log_2 n)$	$O(1)$	$O(1)$
刪除複雜度	$O(log_2 n)$	$O(log_2 n)$	$O(1)$	$O(1)$	$O(log_2 n)$	$O(log_2 n)$	$O(1)$	$O(1)$
搜尋複雜度	$O(log_2 n)$	$O(log_2 n)$	$O(1)$	$O(1)$	$O(log_2 n)$	$O(log_2 n)$	$O(1)$	$O(1)$
次序關係	有	有	無	無	有	有	無	無
迭代器移動	有	有	無	無	有	有	無	無
可重複與否	否	否	否	否	可	可	可	可

unordered：雜湊表，否則為紅黑樹
multi：可重複，否則不可重複

Set & Map 與紅黑樹

當你用 set 或 map 時

心裡要有一棵**紅黑樹**



Example Code

Mission

LeetCode #217. Contains Duplicate

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Ref: https://leetcode.com/problems/contains-duplicate/

Example Code

Mission

There is a street of length x whose positions are numbered 0,1,…,x. Initially there are no traffic lights, but n sets of traffic lights are added to the street one after another. Your task is to calculate the length of the longest passage without traffic lights after each addition.

Ref：https://cses.fi/problemset/task/1163

Example Code

Mission

- Input：The first input line contains two integers x and n: the length of the street and the number of sets of traffic lights. Then, the next line contains n integers p1,p2,...,pn: the position of each set of traffic lights. Each position is distinct.
- Output：Print the length of the longest passage without traffic lights after each addition.

Example Code

Mission

- Example
  - Input:

    8 3

    3 6 2
  - Output：

    5 3 3

Example Code

Example Code

Example Code

Practice

Mission

Try LeetCode #349 Intersection of Two Arrays

Given two arrays, write a function to compute their intersection.

Ref: https://leetcode.com/problems/intersection-of-two-arrays/

Example 1:

```
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2]
```

Example 2:

```
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [9,4]
```

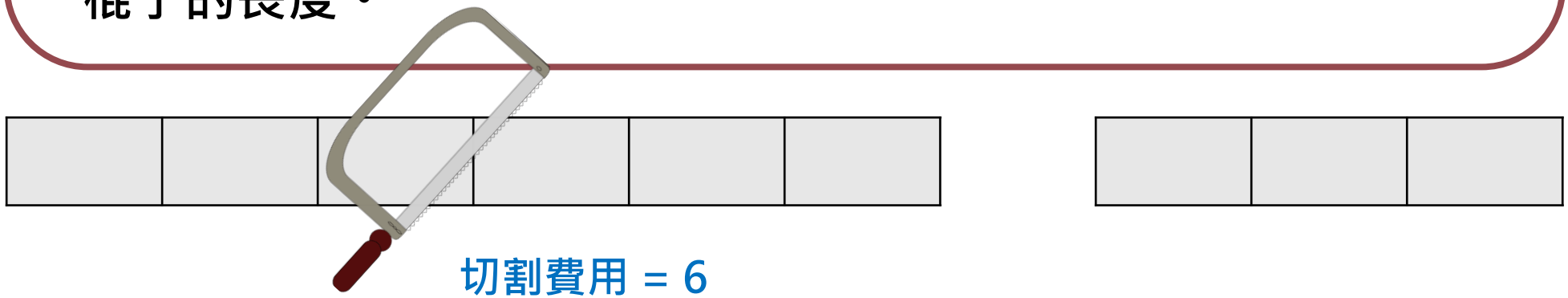
Practice

Mission

APCS 2021/01/09 #3

有一根長度為 L 的木棍，總共會被切割 n 次。

木棍最左端在數線上 0 的位置，右端則在數線上 L 的位置，每次切割都會由一個介於 0 到 L 的數字表示要切割的位置，要把穿過個這位置的棍子切成兩段，而所需的花費就等於所切割的棍子的長度。



切割費用 = 6

Practice

Mission

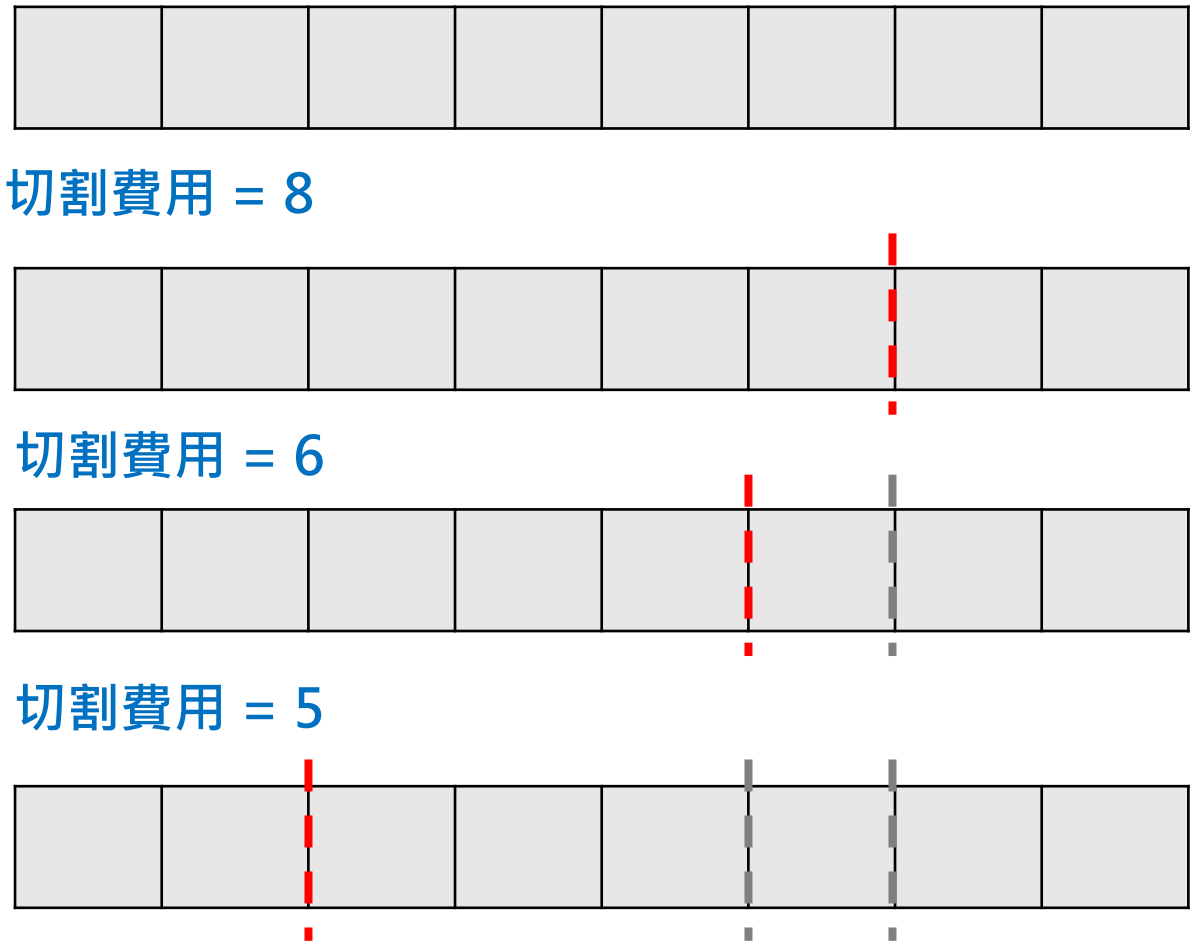
- 輸入：
  1. 第一行為兩個正整數 n,L
     - 代表切 n 次、木棍長 L。
  2. 接下來 n 行每行有兩個整數 p, i
     - 表示 p 位置被切割
     - 而這刀是全部切割中的第 i 刀
     - 且保證 i 介於 [1,n] 的正整數且不會重複。
- 輸入：
  1. 輸出全部的切割費用
     - 答案可能超過 2^{31} 但不會超過 2^{60} 。

Practice

Mission

- 範例輸入：
  3 8
  2 3
  6 1
  5 2
- 範例輸出：
  19
- 說明：
  三次切割各需 8、6、5元。

https://zerojudge.tw/ShowProblem?problemid=f607



Take Home Message

- 集合 (Set) 的常見用途是？
- 併查集 (Disjoint Sets) 與一般集合差在哪？
- 進行 Collapsing 的目的為何？
- 併查集合併集合時要進行何種操作？
- Map 與 Set 差異在哪？
- Set 與 Multiset 差異在哪？