Data Structure I

關於這堂課

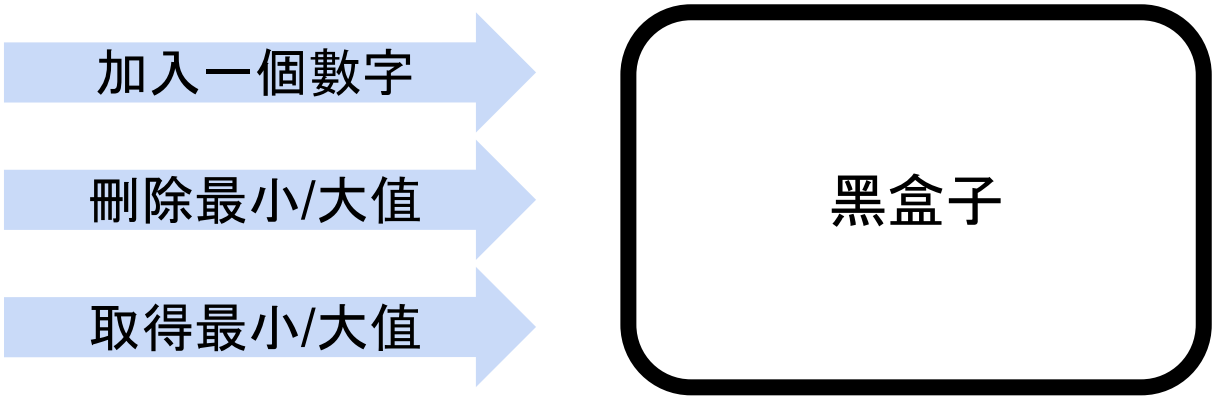
- 先備知識
  - 時間複雜度

- 學習重點
  - 基本資料結構使用與複雜度分析
  - Standard Template Library (STL)

Data Structure

- 序列結構
  - Vector, List, Stack Queue
- 樹狀結構
  - Heap, Set, Map
- Hash
  - Hash Table

抽象資料型態 (ADT)



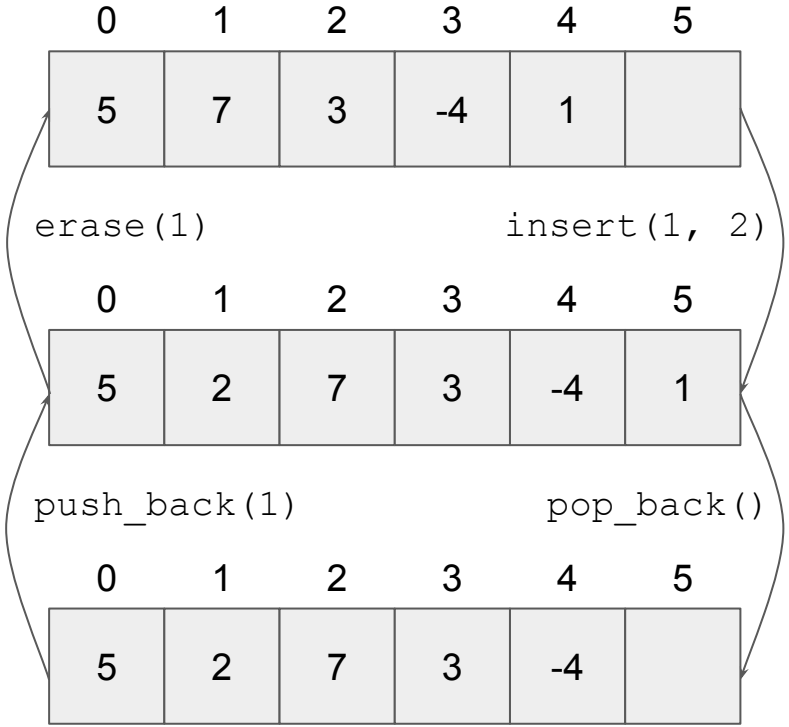
Vector

vector

- 在記憶體中的一段連續空間
- C++ STL 中的 `vector` 或 `array`

vector 操作

- access (operator[])
  - 時間複雜度 O(1)
- insert / erase
  - 時間複雜度 O(n)
- push_back / pop_back
  - 時間複雜度 O(1)
- lower_bound / upper_bound
  - 時間複雜度 O(lgn)



std::vector 使用方法

```
vector<int> vec(5, 0);  // {0, 0, 0, 0, 0}
vec[2] = 4;             // {0, 0, 4, 0, 0}
vec.insert(vec.begin() + 3, 6);
                        // {0, 0, 4, 6, 0, 0}
vec.erase(vec.begin() + 2);
                        // {0, 0, 6, 0, 0}
vec.push_back(3);       // {0, 0, 6, 0, 0, 3}
vec.pop_back();         // {0, 0, 6, 0, 0}
```

zerojudge d323

- 給一個正整數序列長度 n, 排序並輸出
- $1 \le n \le 10^7$

輸入:	輸出:
4 4 3 2 1	1 2 3 4

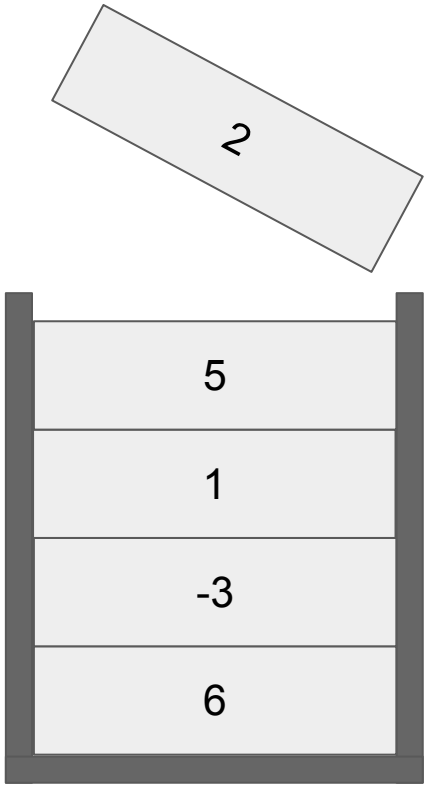
題單 - vector

- ZeroJudge a040 阿母斯壯數字
- ZeroJudge b552: 找質數
- 多數要使用陣列的題目

Stack

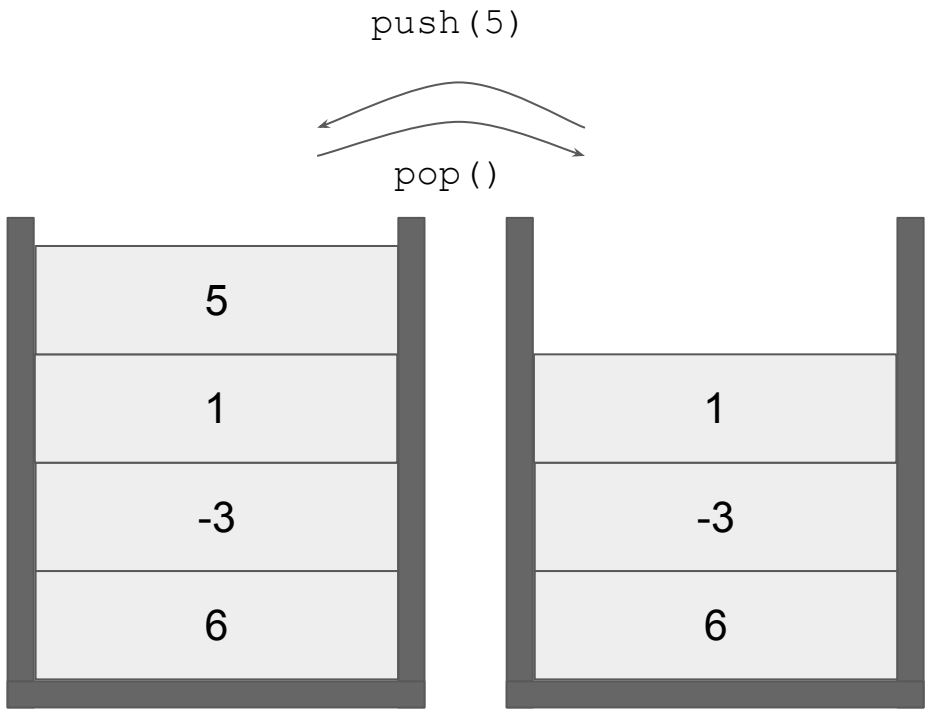
Stack

- 資料堆在一個單開口管子內
  - 只能從開口放入或取出
- 維持 先進後出
  - First In Last Out (FILO)
- C++ STL 中的 `stack`



Stack 操作

- top
  - 時間複雜度 O(1)
- push / pop
  - 時間複雜度 O(1)



std::stack 使用方法

```
stack<int> st;                  // {}

st.push(3);                     // {3}
st.push(4);                     // {3, 4}

st.top();                       // 4

st.pop();                       // {3}
st.pop();                       // {}
```

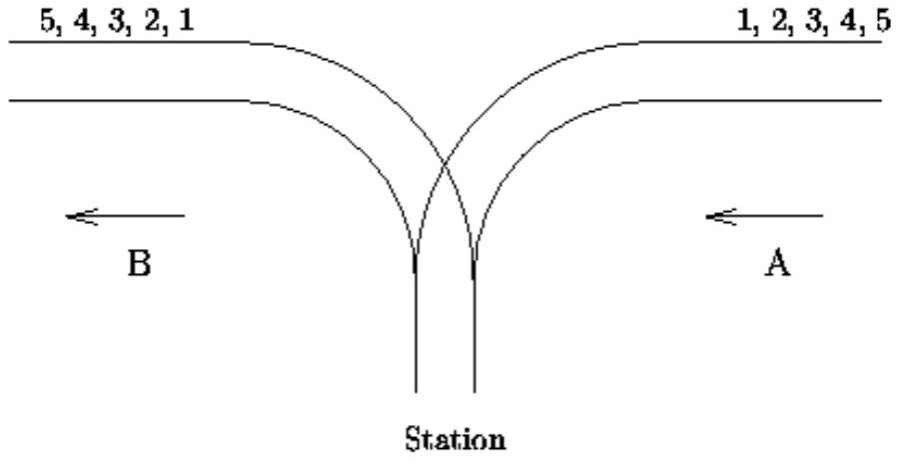
LeetCode 20: Valid Parentheses

- 給一個字串 S, 問是否為合法括號匹配
- $1 \le |S| \le 10^4$, S 包含 (), [], {} 三種括號

```
()[]{}  -> true
(]      -> false
{[]}    -> true
```

zerojudge c123 : Rails

- 給一個序列 B, 問可不可能由 A 造出這種排列
- $1 \le |B| \le 10^3$



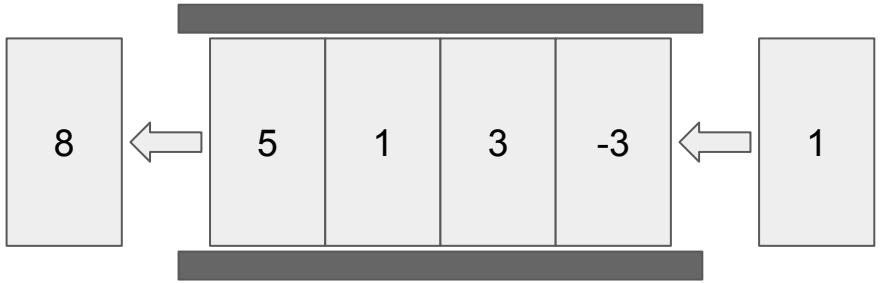
題單 - Stack

- LeetCode 20: Valid Parentheses
- zerojudge c123：Rails
- zerojudge a017：五則運算
- zerojudge c907：尋找最大矩形
- zerojudge a565: 2.p&q的邂逅
- Aizu - ALDS1_3_A: Stack

Queue

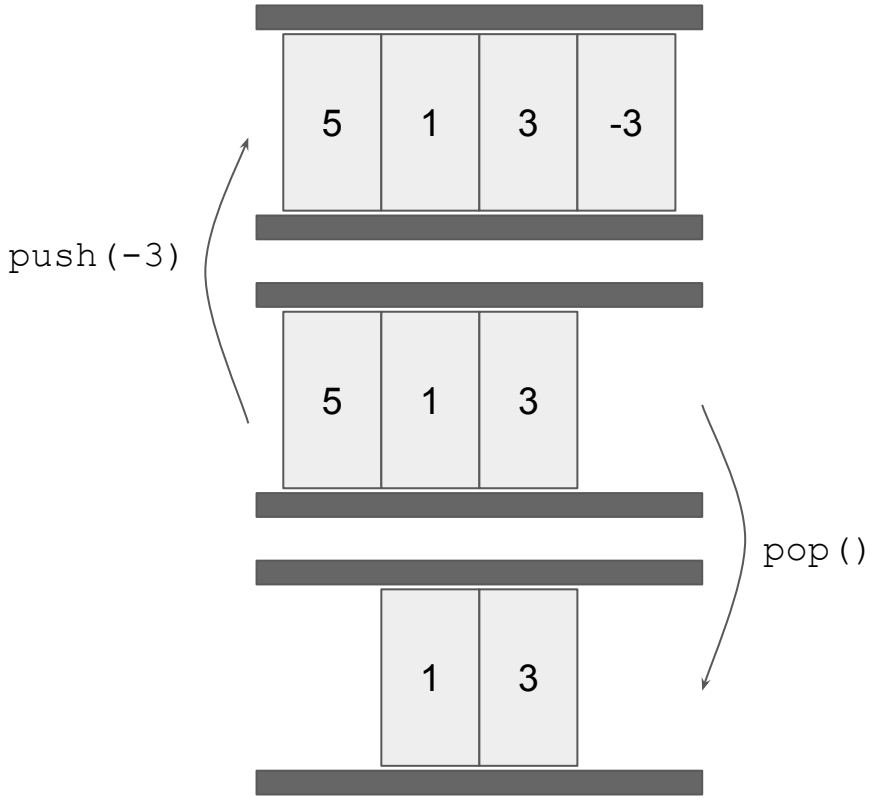
Queue

- 資料堆放在雙開口管子內
  - 只能從前端取出, 從後端放入
- 維持先進先出
  - First In First Out (FIFO)
- C++ STL 中的 `queue`



Queue 操作

- front
  - 時間複雜度 O(1)
- push / pop
  - 時間複雜度 O(1)



std::queue 使用方法

```
queue<int> que;                    // {}

que.push(3);                       // {3}
que.push(4);                       // {3, 4}

que.front();                       // 3

que.pop();                         // {4}
que.pop();                         // {}
```

LeetCode 933. Number of Recent Calls

- 呼叫多次 `ping` function，每次呼叫會給一個時間點 `t`，
  要輸出 `[t - 3000, t]` 內 `ping` function 總共被呼叫了幾次
- 每次給的時間點 `t` 是遞增的

```
ping 1       // [-2999,    1] 總共被呼叫 1 次
ping 100     // [-2900,  100] 總共被呼叫 2 次
ping 3001    // [    1, 3001] 總共被呼叫 3 次
ping 3002    // [    2, 3002] 總共被呼叫 3 次
```

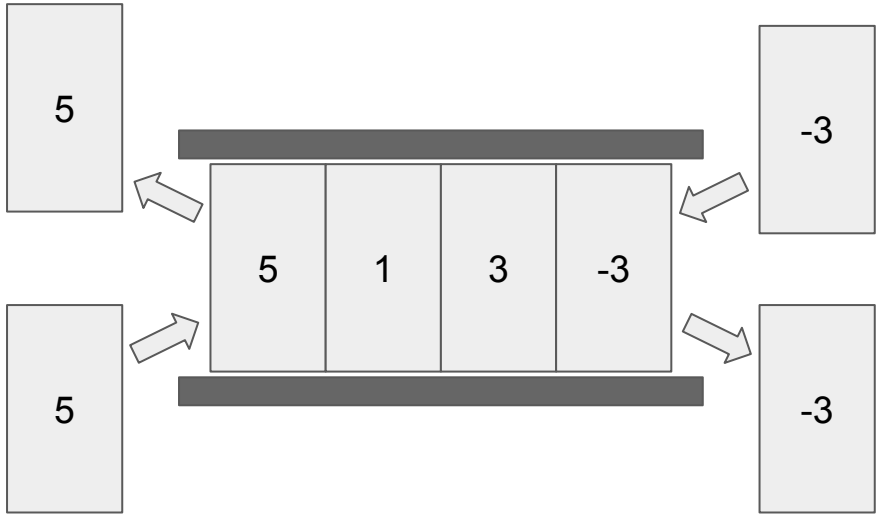
題單 - Queue

- zerojudge e155: 10935 - Throwing cards away I
- LeetCode 933: Number of Recent Calls
- ALDS1_3_B: Queue
- UVa 540: Team Queue
- UVa 10935: Throwing cards away I
- UVa 12100: Printer Queue
- Codeforces 1239C

Deque

Deque [`dek]

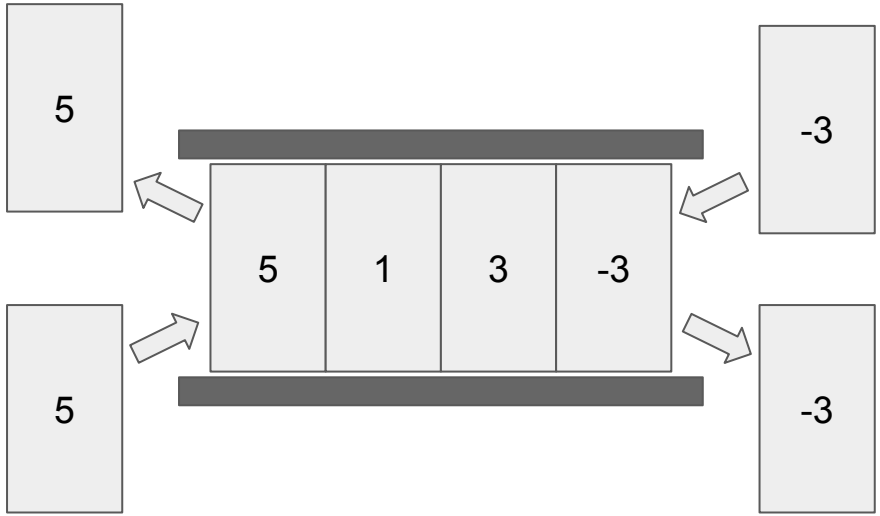
- 資料堆放在兩個開口的管子內
  - 雙向都可以放入或取出
- C++ STL 中的 `deque`



Deque 操作

- front / back
  - 時間複雜度 O(1)

- push_back / pop_back
  push_front / pop_front
  - 時間複雜度 O(1)



std::deque 使用方法

```
deque<int> deq;                 // {}

deq.push_front(3);              // {3}
deq.push_back(4);               // {3, 4}
deq.push_front(5);              // {5, 3, 4}

deq.front();                    // 5
deq.back();                     // 4

deq.pop_back();                 // {5, 3}
deq.pop_front();                // {3}
```

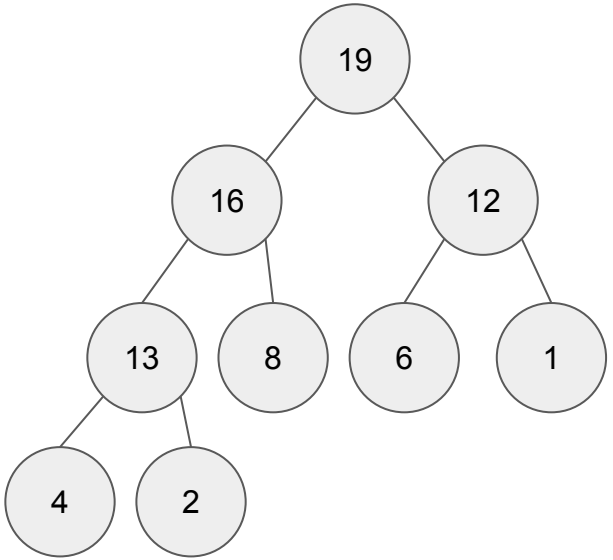
題單 - Deque

- LeetCode 1670. Design Front Middle Back Queue

Heap

Heap

- 資料透過二元樹狀結構堆疊起來
- 最大堆：任一個節點的值比小孩都還大
  - 最大的數字會在最上面
- C++ STL 中的 `priority_queue`



Heap 操作

- top
  - 時間複雜度 O(1)
- push / pop
  - 時間複雜度 O(log n)

std::priority_queue 用法

```
priority_queue<int> pq;

pq.push(4);
pq.push(5);
pq.push(3);

pq.top();                    // 5

pq.pop();

pq.top();                    // 4
```

LeetCode 1845. Seat Reservation Manager

- 實作 SeatManager, reserve 和 unreserve 函式
  - SeatManager(int n): 開放 n 個空座位, 編號 1 到 n
  - int reserve(): 回傳空座位中最小編號, 並變成非空座位
  - void unreserve(int idx): 將編號為 idx 的非空座位變成空座位

LeetCode 703. K^{th} Largest Element in a Stream

- 給一個陣列跟 `k`
- 每次讓這個陣列增加一個 element，增加完後要輸出當前第 k^{th} 大的 element

題單 - Heap

- zerojudge d129: 00136 - Ugly Numbers
- zerojudge c122: 00443 - Humble Numbers
- LeetCode 1845. Seat Reservation Manager
- LeetCode 703. Kth Largest Element in a Stream
- ALDS1_9_B: Maximum Heap
- ALDS1_9_C: Priority Queue

Map / Set

Set / Map

- 資料以二元樹狀結構存取
  - 二元搜尋樹 (Tree I. Binary Search Tree 詳述)

- C++ STL 中的 `set / map`

Set / Map 操作

- begin / end
  - 時間複雑度 O(lgn)
- find / count
  - 時間複雑度 O(lgn)
- insert / erase
  - 時間複雑度 O(lgn)
- lower_bound / upper_bound
  - 時間複雑度 O(lgn)

std::set 使用方法

```
set<int> st;

st.insert(4);                    // {4}
st.insert(6);                    // {4, 6}
st.insert(4);                    // {4, 6}

*st.begin();                     // 4
st.count(6);                     // 1
st.count(5);                     // 0
st.erase(4);                     // {6}
```

std::set 輸出每個元素

```
set<int> st = {1, 3, 5}

// c++ 11
for (int x : st) {
    cout << x << '\n';
}
```

std::set 輸出每個元素

```
set<int> st = {1, 3, 5}

// c++ 98
for (set<int>::iterator it = st.begin();
      it != st.end(); it++) {
    cout << x << '\n';
}
```

std::set 找最大最小值

```
set<int> st = {1, 3, 5}

cout << "min: " << *st.begin() << '\n';
cout << "max: " << *st.rbegin() << '\n';
```

std::map 使用方法

```
map<string, int> mp;

mp[“seacow”] = 5;                    // { (“seacow”, 5) }

mp.count(“seacow”);                  // 1
mp.count(“hello”);                   // 0

mp.erase(“seacow”);                  // {}
```

std::map 輸出每個元素

```
map<string, int> mp;

mp[“Alice”] = 5;
mp[“Bob”] = 10;
mp[“Carol”] = 7;

// c++ 17
for (auto [name, val]: mp){
    cout << name << ‘ ‘ << val << ‘\n’;
}
```

std::map 輸出每個元素

```
map<string, int> mp;

mp[“Alice”] = 5;
mp[“Bob”] = 10;
mp[“Carol”] = 7;

// c++ 11
for (auto p: mp){
    cout << p.first << ‘ ‘ << p.second << ‘\n’;
}
```

std::map 輸出每個元素

```
map<string, int> mp;

mp[“Alice”] = 5;
mp[“Bob”] = 10;
mp[“Carol”] = 7;

// c++ 98
for (map<string, int> iterator it = mp.begin();
      it != mp.end(); it++) {
    cout << it->first << ‘ ’ << it->second << ‘\n’;
}
```

LeetCode 217. Contains Duplicate

- 給一個序列 nums, 問裡面的所有元素是否均出現至少兩次。
- $1 \le |nums| \le 10^5$, $-10^9 \le nums_i \le 10^9$

```
nums = [1,2,3,4]                -> false
nums = [1,1,1,3,3,4,3,2,4,2]    -> true
```

題單 - Set / Map

- zerojudge d194: 11572 - Unique Snowflakes (相異個數)
- zerojudge d442: 10591 - Happy Number (紀錄是否出現過)
- zerojudge b523: 先別管這個了，你聽過安麗嗎？
- zerojudge d244: 一堆石頭
- zerojudge b162: NOIP2007 1.统计数字 (列出所有東西)
- zerojudge c421: pA 雲端列印 (使用 begin(), rbegin() 找極值)

題單 - Set / Map

- LeetCode 217. Contains Duplicate
- LeetCode 532. K-diff Pairs in an Array
- LeetCode 1590. Make Sum Divisible by P
- Codeforces 975C
- Codeforces 978F
- Codeforces 1077E

Hash Table

Hash Table

- 使用 hash function

- C++ STL 中的 `unordered_set / unordered_map`
  - 用法和 `set / map` 相似

Hash Table 操作

- find / count
  - 時間複雑度 O(1)
- insert / erase
  - 時間複雑度 O(1)

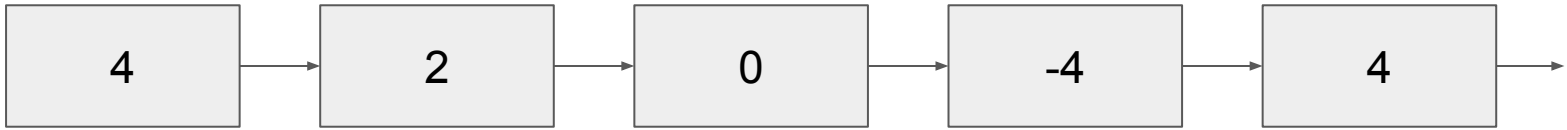
題單 - Hash Table

- LeetCode 1. Two Sum
- LeetCode 1512 Number of Food Pairs
- LeetCode 895. Maximum Frequency Stack
- LeetCode 454. 4Sum II

Linked List

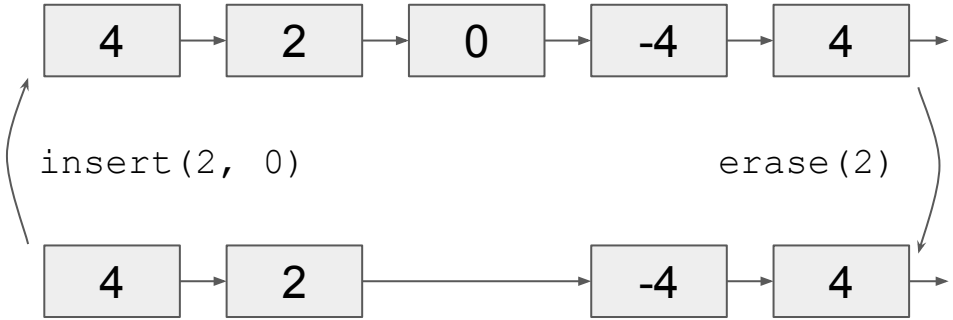
Linked List

- 在記憶體中不連續, 使用指標連接



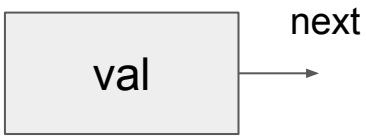
Linked List 操作

- access / find
  - 時間複雜度 O(n)
- insert / erase
  - 時間複雜度 O(1)



Linked List 實作 - ListNode

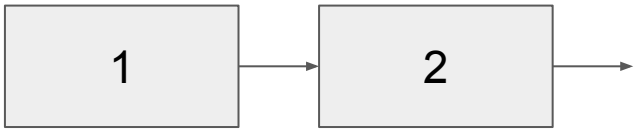
```
struct ListNode {
  int val;
  ListNode* next;
};
```



Linked List 實作 - ListNode

```
ListNode *node1 = new ListNode();
node1->val = 1;

node->next = new ListNode();
node->next->val = 2;
```



LeetCode 141. Linked List Cycle

- 給一個 List, 問有沒有環

題單 - Linked List

- LeetCode 206. Reverse Linked List
- LeetCode 21. Merge Two Sorted Lists
- LeetCode 83. Remove Duplicates from Sorted List
- LeetCode 160. Intersection of Two Linked Lists
- LeetCode 234. Palindrome Linked List
- UVA 11988 Broken Keyboard

More Practices about Data Structure

vjudge 題單：https://vjudge.net/contest/418791