Chapter 6
排序與搜尋
Sorting and Searching

台南女中資訊研究社 38th C++ 進階班課程

TNGS IRC 38th C++ Advanced Course

講師：陳俊安 Colten

這一個單元的重點

- 二分搜尋 Binary Search
- 雙指針 Two Pointers
- 滑動窗口 Sliding Window

二分搜尋 Binary Search

- 一種有效率的搜尋方法
- 可以算是演算法中，最具有代表性的一個搜尋法
- 通常我們簡稱二分搜

二分搜尋 Binary Search

- 有一個介於 1 ~ 10^9 之間的幸運數字，你必須找到這個數字是多少
- 猜錯會告訴你，你猜得太小還是太大
- 請問你最多只需要花幾次一定可以猜到？

二分搜尋 Binary Search

- 策略應該很多人都知道，每一次都猜中間的數字
  - 可以讓數字範圍減小一半
  - 這個就是二分搜的精神
  - 每一次都讓我們需要搜尋的範圍減少一半
  - 這樣子我們最多只需要花費 log10^9 + 1 次即可猜中

二分搜尋 Binary Search

- 如果把二分搜寫成程式？
- 有一句很好的名言：
  - 二分搜概念簡單，但其細節令人難以招架...
  - Why？

二分搜的常犯錯誤

- long long 的問題
- overflow
- 最後的答案？
- 單調性

二分搜的常犯錯誤 - long long 的問題

- 假設現在我們要尋找的範圍是區間 [L , R]
- 寫成程式：(L + R) / 2
- (L + R) 這個部分如果會超出 int 範圍，就必須記得把 l , r 開成 long long 的型態

二分搜的常犯錯誤 - overflow的問題

- 假設現在我們要尋找的範圍是區間 [L , R]
- 寫成程式：(L + R) / 2
- 如果 R 太大，有可能會發生連 long long 甚至 unsigned long long 都塞不下的情況，因此一開始的 R 要設多少要特別注意

二分搜的常犯錯誤 - 最後的答案？

- 二分搜 10 個人會有 10 種不同的寫法
- 有些人的寫法最後 L，R 會夾在一起，而這個位置就是答案
- 但是有些人的寫法最後 L，R 不會夾在一起（像是我習慣的寫法）
  - 這個時候就必須判斷答案到底是 L 還是 R 了
  - 建議大家在摸索的過程中找出一個自己最習慣的寫法

二分搜的常犯錯誤 - 最後的答案？

- 我二分搜的寫法，假設要縮邊界：
  - 縮左界：L = mid + 1
  - 縮右界：R = mid - 1
- 我這種寫法就會發生最後 L , R 不會重疊的情況
- 因此最後怎麼判斷答案是 L 還是 R 就變成了一個重要問題
- 等等題目實際的例子會告訴大家我怎麼判斷 L , R 的
- 不過大家不一定要學我的二分搜寫法，自己習慣就好

二分搜的常犯錯誤 - 最後的答案？

- 二分搜有一個重點是，判斷完 mid 必須有辦法判斷答案皆下來應該會落在左半部還是右半部，這個性質我們稱為 單調性
- 如果沒有單調性是沒有辦法進行二分搜的
  - 因為無法判斷答案接下來會落在左半部還是右半部

一些二分搜的好用 Function

- lower_bound(L , R , value) // O(logN)
- upper_bound(L , R , value) // O(logN)
- binary_search(L , R , value) // O(log N)
- 使用之前序列必須排序！！！

lower_bound(L , R , value)

- 找到 [L , R) 第一個 >= value 的指標 or 迭代器位置
  - 如果 L , R 是迭代器，回傳的就會是迭代器
  - 如果 L , R 是指標，回傳的就會是指標
  - 如果找不到滿足的，會回傳 R

```
21      vector<int>a(10);
22      int b[5] = {1,3,5,7,9};
23
24      for(int i=0;i<10;i++) a[i] = i;
25
26      cout << *lower_bound(a.begin(),a.end(),3) << "\n"; // 3
27      cout << *lower_bound(b,b+5,6) << "\n"; // 7
```

upper_bound(L , R , value)

- 找到 [L , R) 第一個 > value 的指標 or 迭代器位置
  - 如果 L , R 是迭代器，回傳的就會是迭代器
  - 如果 L , R 是指標，回傳的就會是指標
  - 如果找不到滿足的，會回傳 R

```
21      vector<int>a(10);
22      int b[5] = {1,3,5,7,9};
23
24      for(int i=0;i<10;i++) a[i] = i;
25
26      cout << *upper_bound(a.begin(),a.end(),3) << "\n"; // 4
27      cout << *upper_bound(b,b+5,6) << "\n"; // 7
```

binary_search(L , R , value)

- 找看看 [L , R) 是否存在 value
  - 有的話會回傳 true
  - 沒有的話會回傳 false

```
20
21      int b[5] = {1,3,5,7,9};
22
23      cout << binary_search(b,b+5,6) << "\n"; // 0
24      cout << binary_search(b,b+5,7) << "\n"; // 1
```

set 與 map 上使用這些工具

- set 與 map 背後實作原理比較特殊
- 如果直接使用一般的 lower_bound 那些工具，時間複雜度會退化成 O(N)
- 不過不用擔心，set 與 map 有內建自己專屬的工具
  - .lower_bound
  - .upper_bound
  - 沒有 binary_search，要用就用 find 就好

set 與 map 上使用這些工具

- 如果想要在 set 與 map 上使用這些工具，必須把他當成跟 STL 的指令來使用，請看下圖的例子，這樣時間複雜度才會是預期的 O(logN)
- 特別注意的是，在 map 與 set 上使用時，無法指定搜尋的區間

```
21      set <int> s;
22      for(int i=1;i<=5;i++) s.insert(i);
23
24      cout << *s.lower_bound(3) << "\n"; // 3
25      cout << *s.upper_bound(3) << "\n"; // 4
```

今天ㄉ小測驗

- 給定一個長度 n（n <= 10^5）的序列
- 接下來有 Q 組查詢，每一組查詢給一個數字 x
- 請你針對每一組查詢輸出有幾個數字 <= x
- 你的程式必須在 1 秒內跑完