

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 $1 \sim 10^9$ 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 $\log_{10} 9 + 1$ 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，
必須記得把 l, r 開成 long long 的型態

就

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： $L = \text{mid} + 1$
 - 縮右界： $R = \text{mid} - 1$
- 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) // $O(\log N)$`
- `upper_bound(L , R , value) // $O(\log N)$`
- `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 \geq 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

```
20
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
28 }
```

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 $O(\log N)$
- 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小練習

- 給定一個長度 n ($n \leq 10^5$) 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 $\leq x$
- 你的程式必須在 1 秒內跑完

一些二分搜的好用 Function 的補充

- 當使用 pair 時，記得會優先比較 first 再比較 second

```
12  vector <pair<long long,long long>> a;
13
14  for(int i=1;i≤5;i++) a.push_back(make_pair(i,1));
15
16  // 上面那一行也可以寫成 a.push_back({i,1}); [C++ 11 以上]
17
18  sort(a.begin(),a.end());
19
20  auto it = lower_bound(a.begin(),a.end(),make_pair(3LL,2LL));
21
22  // 3LL 會使 3 是 long long 的型態 且 上面這行不能寫成 {3,2}
23
24  cout << (*it).first << " " << (*it).second << "\n"; // 4 1
25
26  // 一定要像這樣連 * 都括號起來，不然會發生衝突
```

一些二分搜的好用 Function 的補充

- 如果你想要使用你自己自定義的規則排序，當然可以

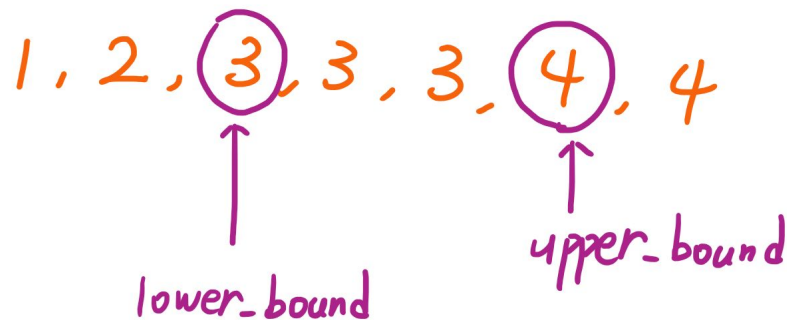
```
7 bool cmp(pair<int,int> a,pair<int,int> b)
8 {
9     if( a.second == b.second ) return a.first < b.first;
10
11     return a.second < b.second;
12 }
```

```
18 vector <pair<long long,long long>> a;
19
20 for(int i=1;i≤5;i++) a.push_back(make_pair(i,1));
21
22 // 上面那一行也可以寫成 a.push_back({i,1}); [C++ 11 以上]
23
24 sort(a.begin(),a.end(),cmp);
25
26 auto it = lower_bound(a.begin(),a.end(),make_pair(3LL,1LL),cmp);
27
28 cout << (*it).first << " " << (*it).second << "\n"; // 3 1
29
30 // 一定要像這樣連 * 都括號起來，不然會發生衝突
```

例題：TOJ 55 Numbers

upper_bound 與 lower_bound 的搭配

試著去觀察 迭代器 位置



Ans: upper - lower

關鍵程式碼

```
36         int input;
37
38         cin >> input;
39
40         auto i1 = upper_bound(a.begin(),a.end(),input);
41
42         auto i2 = lower_bound(a.begin(),a.end(),input);
43
44         cout << i1 - i2 << "\n";
```

經典技巧：[APCS 2017 3 月 第四題 基地台](#)

經典技巧：對答案二分搜

APCS 2017 3 月第四題基地台

現在有 N ($1 \leq N \leq 50000$) 個服務點，給你這 N 個服務點的座標，你現在可以隨意在任何地方架設 K 個基地台，請問基地台的服務直徑最少只需要多少服務範圍就可以到所有服務點（所有服務點都是在一條數線上），座標範圍最多到 10^9

如果我給你一個直徑，你能回答這一個直徑是不是 ok 的嗎？

經典技巧：APCS 2017 3 月 第四題 基地台

- 假設我現在請你看看 x 長度的直徑可不可以滿足題目要求
 - 如果可以 \rightarrow 最終的答案一定會 $\leq x$
 - 如果不行 \rightarrow 最終的答案一定會 $> x$
 - 有了這一個性質 (單調性)，我們可以使用二分搜

經典技巧：APCS 2017 3 月 第四題 基地台

- 怎麼看 x 這一個長度可不可以？
 - 如果現在第一個人的位置在 p
 - 你的第一個基地台怎麼放效果最好？
 - 讓這個基地台的範圍剛好是 $p \sim p + x$
 - 如果有人的位置 $> p + x$ ，表示需要多一台基地台
 - 如此一來我們就可以用這樣的概念去看每一次二分搜的長度有沒有滿足條件了

實作：先排序服務點的位置

```
15     vector <int> a;  
16  
17     cin >> n >> k;  
18  
19     for(int i=0;i<n;i++)  
20     {  
21         int input;  
22  
23         cin >> input;  
24  
25         a.emplace_back(input);  
26     }  
27  
28     sort(a.begin(),a.end());
```

設定好 左界 跟 右界

怕自己右界設錯的人可以故意稍微開大一點點比較保險

```
29  
30     int l = 0, r = ( a[n-1] - a[0] ) / k + 1;  
31  
32     int ans = 3e9;
```

取 mid 並看看是否是 ok 的，並縮小答案區間

```
int mid = ( l + r ) / 2;
```

```
40     int now = a[0] + mid; // 第一個基地台可以涵蓋到的位置
41
42     int u = 1; // 需要的基地台數量
43
44     for(int i=1;i<n;i++)
45     {
46         if( a[i] ≤ now ) continue; // 已經被當前最新的基地台涵蓋到了
47
48         else // 當前基地台都涵蓋不到，所以要架一個新的
49         {
50             u++;
51
52             now = a[i] + mid; // 新的基地台可以涵蓋到的範圍
53         }
54     }
```

```
if( u ≤ k ) r = mid - 1;
else l = mid + 1;
```

雙指針 Two Pointers

- 顧名思義，用兩個指針維護狀態的技巧
- 雙指針是一個優化枚舉的技巧

範例：選數字

- 給定兩組長度 N 的序列 a , b 與一個正整數 x
- 請找出有多少 (i,j) 滿足 $a_i + b_j \geq x$
- 大家應該都能想到 $O(N^2)$ 的作法
 - 枚舉
- 其實可以做到 $O(N \log N)$ 哦

範例：選數字

- 如果我們先把兩個序列都由小到大排序？
- 試著想想看，如果 $a_1 + b_5 \geq x$ 代表什麼？
 - $a_1 + b_5 \geq x$
 - $a_2 + b_5 \geq x$
 - $a_3 + b_5 \geq x$
 - and so on...
- 因此對 $a_2 \sim a_n$ 來說，沒有必要去枚舉 $b_5 \sim b_n$

範例：選數字

- 用這樣的概念來看，我們可以把程式碼寫成：

```
28
29     sort(a.begin(),a.end());
30     sort(b.begin(),b.end())
31
32     int idx = n , ans = 0;
33
34     for(int i=1;i<=n;i++)
35     {
36         while( idx >= 1 && a[i] + b[idx] >= x ) idx--;
37         ans += ( n - idx );
38     }
39
40     cout << ans << "\n";
```

範例：選數字

- 枚舉的時間複雜度 $O(n)$ ，排序 $O(n \log n)$

```
28
29     sort(a.begin(),a.end());
30     sort(b.begin(),b.end())
31
32     int idx = n , ans = 0;
33
34     for(int i=1;i<=n;i++)
35     {
36         while( idx >= 1 && a[i] + b[idx] >= x ) idx--;
37         ans += ( n - idx );
38     }
39
40     cout << ans << "\n";
```

範例：選數字

- 你會發現 i 是一個指針， idx 也是一個指針，因此這樣的技巧被稱之為雙指針 Two Pointers

```
28
29     sort(a.begin(),a.end());
30     sort(b.begin(),b.end())
31
32     int idx = n , ans = 0;
33
34     for(int i=1;i<=n;i++)
35     {
36         while( idx >= 1 && a[i] + b[idx] >= x ) idx--;
37         ans += ( n - idx );
38     }
39
40     cout << ans << "\n";
```

範例：CSES Sum of Two Values

Sum of Two Values

給定一個長度為 n ($1 \leq n \leq 2 \times 10^5$) 的序列，請你找到兩個不同的位置 i, j ，滿足 $a_i + a_j = x$ ，如果找不到則輸出 IMPOSSIBLE

這題也可以用 map 或二分搜做掉

範例：CSES Sum of Two Values

Sum of Two Values

給定一個長度為 n ($1 \leq n \leq 2 \times 10^5$) 的序列，請你找到兩個不同的位置 i, j ，滿足 $a_i + a_j = x$ ，如果找不到則輸出 IMPOSSIBLE

- 我們一樣用枚舉的方式來做
- 先將整個序列由小到大排序好

範例：CSES Sum of Two Values

Sum of Two Values

給定一個長度為 n ($1 \leq n \leq 2 \times 10^5$) 的序列，請你找到兩個不同的位置 i, j ，滿足 $a_i + a_j = x$ ，如果找不到則輸出 IMPOSSIBLE

- 我們一樣用枚舉的方式來做
- 先將整個序列由小到大排序好
- 去找當前排序後的 a_1 可以跟哪一個元素配
- 檢查 $a_1 + a_n$ 的結果，如果 $a_1 + a_n > x$ 那我們就去檢查 $a_1 + a_{n-1}$
- 因為 $a_1 + a_n > x$ 那麼 $a_2 + a_n$ 的結果一定也大於 x

範例：CSES Sum of Two Values

Sum of Two Values

給定一個長度為 n ($1 \leq n \leq 2 \times 10^5$) 的序列，請你找到兩個不同的位置 i, j ，滿足 $a_i + a_j = x$ ，如果找不到則輸出 IMPOSSIBLE

- 我們一樣用枚舉的方式來做
- 先將整個序列由小到大排序好
- 去找當前排序後的 a_1 可以跟哪一個元素配
- 檢查 $a_1 + a_n$ 的結果，如果 $a_1 + a_n > x$ 那我們就去檢查 $a_1 + a_{n-1}$
- 因為 $a_1 + a_n > x$ 那麼 $a_2 + a_n$ 的結果一定也大於 x
- 如果 $a_1 + a_n < x$ 接下來就直接檢查 $a_2 + a_n$
- 因為 $a_1 + a_n < x$ ，那麼 $a_1 + a_{n-1}$ 也一定小於 x

範例：CSES Sum of Two Values

CSES Sum of Two Values

給定一個長度為 n ($1 \leq n \leq 2 \times 10^5$) 的序列，請你找到兩個不同的位置 i, j ，滿足 $a_i + a_j = x$ ，如果找不到則輸出 IMPOSSIBLE

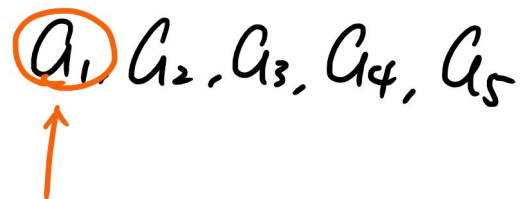
- 照這樣的規則下去，你會發現我們就是在控制兩個指針
- 當第一個指針的元素加上第二個指針的元素 $> m$ 時，移動第二個指針，讓總和變小一點， $< m$ 時移動第一個指針，讓總和變大一點
- 這樣兩個指針一直往內縮，直到找到答案為止
- 時間複雜度 $O(n + n \log n)$

雙指針經典題：Sum of Three Values

雙指針的經典題，究竟應該要如何使用雙指針？

枚舉其中一個位置

a_1, a_2, a_3, a_4, a_5



- $a + b + c = x$
- 假設我枚舉 a , 表示 $b + c = x - a$
- 我必須去找看看 a 右邊的序列有沒有兩個數字相加 $= x - a$
 - Sum of Two Values
- 因此這題的作法其實就只是 Sum of Two Values 外多一層迴圈而已

輸入資料

```
15     int n,m;
16
17     cin >> n >> m;
18
19     vector <pair<int,int>> a(n);
20
21     for(int i=0;i<n;i++)
22     {
23         cin >> a[i].first;
24
25         a[i].second = i + 1; // 紀錄原本的索引值才不會因為排序不知道索引值
26     }
```

排序後雙指針的結果才會是正確的

```
28      sort(a.begin(), a.end());
```

枚舉兩個指針的位置

```
30     for(int i=0;i<n;i++) // 第一個指針
31     {
32         int index = n - 1; // 第三個指針
33
34         for(int k=i+1;k<n;k++) // 第二個指針
```

開始雙指針

```
30     for(int i=0;i<n;i++) // 第一個指針
31     {
32         int index = n - 1; // 第三個指針
33
34         for(int k=i+1;k<n;k++) // 第二個指針
35         {
36             while( index > k ) // 如果第三個指針在第二個指針左邊就不用繼續找了，因為如果有答案早就在之前枚舉到了
37             {
38                 if( a[i].first + a[k].first + a[index].first > m ) index--; // 移動第三個指針
39
40                 else if( a[i].first + a[k].first + a[index].first < m ) break; // 移動第二個指針
41
42                 else // 找到答案了
43                 {
44                     cout << a[i].second << " " << a[k].second << " " << a[index].second << "\n";
45
46                     exit(0);
47                 }
48             }
49         }
50     }
51
52     cout << "IMPOSSIBLE\n"; // 找不到答案 QQ
```

例題：Ten Point Round 一週年紀念賽 Colten 與風原的餅乾大戰爭

這題是一個難以想像可以用雙指針的題目

~~我當初剛好在吃餅乾所以就出了這一題~~

這題的觀察

- 餅乾的價錢很好計算，不是 $G1$ 元就是 $G2$ 元
 - 題目要求只要超過或剛好等於滿足度即可，不用剛剛好
- 既然這樣，同個價格的餅乾我們當然拿滿足度越大的越好

枚舉我們拿了多少個第一排的餅乾

設定一個指針，假設指針在第 i 個位置表示：

我們拿了第一排 $[1, i]$ 區間內的所有餅乾

一開始將第一個指針放在最左邊的位置

設定第二個指針

設定第二個指針， 假設指針在第 i 個位置表示：

我們拿了第二排 $[1, i]$ 區間內的所有餅乾

一開始我們將第二個指針放在最右邊的位置

為什麼第一個在最左邊，第二個在最右邊？

我們可以思考看看，假設我拿了第一排其中 K 個餅乾

那我必須至少拿第二排其中 T 個餅乾才能滿足題目要求

那麼現在假設我拿了第一排其中 $K + 1$ 個餅乾

那我必須至少拿第二排其中 Q 個餅乾才能滿足題目要求

T 跟 Q 的關係會是什麼？

回想一下我們剛剛觀察到的

如果你想要在某一排拿 P 個餅乾

那麼我們當然盡可能拿這一排滿足度大的餅乾

第一排拿的餅乾數量從 K 個 變成 $K + 1$ 個 \cdots 越拿越多時

意味著如果我們如果在第一排拿 K 個餅乾的時候需要拿 Q 個第二排的餅乾補足滿足度

那在我們在第一排拿 $K + 1$ 個餅乾時，會需要第二排的餅乾數量一定不會大於 Q

基於以上性質你會發現

當第一排的指針在往向右移動的時候

第二排的指針也會隨之往左

因此我們就可以利用這樣的遞增性與遞減性來使用雙指針解決這一題

我們其實跟暴力一樣是在枚舉第一排要拿幾個餅乾、第二排要拿幾個餅乾

只是因為題目存在遞增跟遞減的性質，讓我們可以優化我們的枚舉

輸入處理，記得排序

```
13     int n,g1,g2,t;
14
15     cin >> n >> g1 >> g2 >> t;
16
17     vector<int> a(n),b(n);
18
19     int a_total = 0,b_total = 0;
20
21     for(int i=0;i<n;i++)
22     {
23         cin >> a[i];
24     }
25     for(int i=0;i<n;i++)
26     {
27         cin >> b[i];
28
29         b_total += b[i];
30     }
31
32     sort(a.rbegin(),a.rend()); // 會將 a 由大排到小
33
34     sort(b.rbegin(),b.rend()); // 會將 b 由大排到小
```

指針設定

```
36     int ans = 8e9, index = n; // 第二個指針 ( 拿了 index 個第二排的餅乾 )
37
38     for(int i=0; i≤n; i++) // 第一個指針 ( 拿了 i 個第一排的餅乾 )
```

枚舉答案

```
35
36     int ans = 8e9, index = n; // 第二個指針 ( 拿了 index 個第二排的餅乾 )
37
38     for(int i=0; i≤n; i++) // 第一個指針 ( 拿了 i 個第一排的餅乾 )
39     {
40         /* 如果當拿了 i 個第一排的餅乾時，需要 index 個第二排的餅乾，滿足度才會 ≥ t */
41
42         while( a_total + b_total ≥ t && index ≥ 0 )
43         {
44             // 如果當前滿足度 ≥ t，那麼試著減少第二排所拿的數量
45
46             ans = min( i * g1 + index * g2, ans ) // 這次的花費
47
48             if( index - 1 ≥ 0 ) b_total -= b[index-1]; // 更新當前第二排所拿的滿足度總和 ( 注意索引值邊界 )
49
50             index--; // 第二個指針向左移動一格
51         }
52
53         if( i < n ) a_total += a[i]; // 更新當前第一排所拿的滿足度總和
54     }
55
56     if( ans == 8e9 ) cout << -1 << "\n";
57
58     else cout << ans << "\n";
```


滑窗：Sliding Window

- 雙指針延伸出來的一個技巧，整個過程就很像一個區間在往右移動
- 所以其實叫他 Two Pointers 也是可以的

Sliding Window 例題：Subarray Distinct Values

給定一組序列，詢問有多少組區間滿足以下條件：

不相同的元素數量不超過 K 個

如果現在區間 $[3, 5]$ 不相同的元素個數是 $\leq K$ 的

那我們可以確定的是 $[3, 3]$, $[4, 4]$, $[5, 5]$, $[3, 4]$, $[4, 5]$

這幾個區間不相同的元素個數也一定 $\leq K$

當區間 $[L, R]$ 已經超過 K 個不同的元素了

那麼我們可以確定的事情是：

如果區間是 $[L, R']$ 且 $R' > R$ 的話，不同的元素個數也一定超過 K 個

我們利用之前學過的雙指針來進行枚舉

我們先將第一個與第二個指針設置在第一個位置

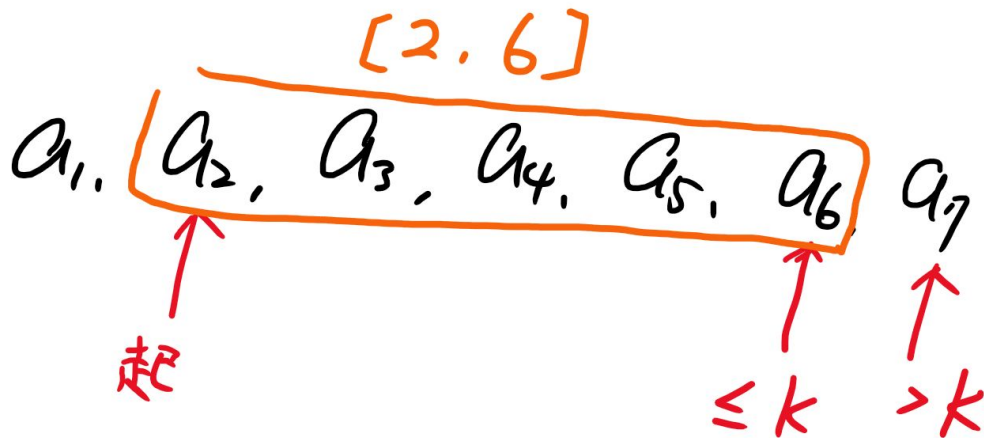
接著，第二個指針開始向右移動，並且我們在移動的時候順便紀錄目前第一個指針與第二個指針這一段區間內不同的元素數量有幾個

當區間內不同的元素數量超過 K 個的時候我們就更新答案

當區間內不同的元素數量超過 K 個的時候我們就更新答案

換句話說，我們很像在枚舉每一個區間的起點，並且盡可能的擴大區間

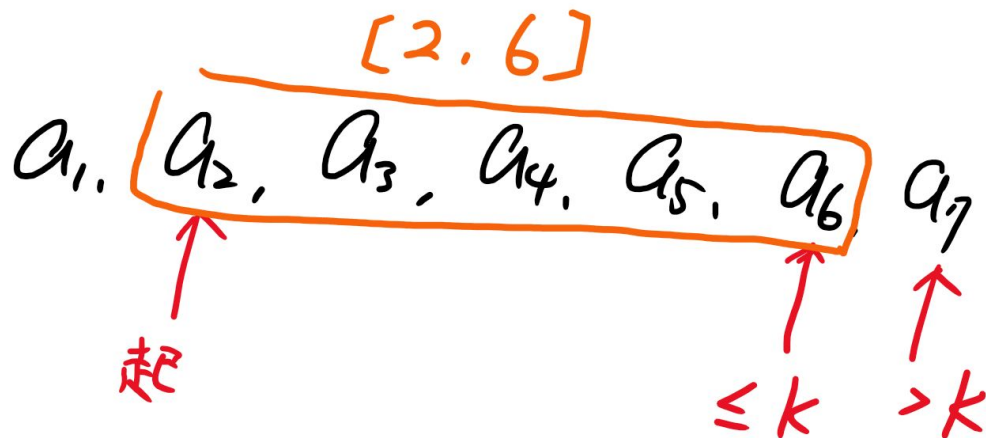
當找到以這個起點當作左界的最長區間之後，對答案做更新



更新答案

以 2 當作起點的答案數量為 5

$[2, 2], [2, 3], [2, 4], [2, 5], [2, 6]$



更新完答案之後就把第一個指針往右移動一步

這樣子一直持續下去就可以把答案計算出來了！

這一題實作的細節很多，讓我們一起來看一下程式碼吧

輸入處理

```
11  
12     int n,k;  
13  
14     cin >> n >> k;  
15  
16     vector <int> a(n);  
17  
18     for(int i=0;i<n;i++) cin >> a[i];
```

一些變數的設定

```
20     int l = 0 , r = 0 , total = 0 , ans = 0;  
21  
22     // l , r 依序代表兩個指針、total 表示 [l,r] 這個區間有幾個不同的元素、ans 紀錄答案
```

利用資料結構維護狀態

```
23  
24     deque <int> dq; // 開一個 dq 來維護我們區間內的所有元素  
25  
26     map <int,int> mp; // 用 map 來維護我們每一個元素在當前區間出現的次數  
27  
28     // 用 map 是因為值域大到 10 的 9 次方，一般陣列無法標記
```

開始進行 Sliding Window

```
29
30     while( r < n )
31     {
32         dq.push_back(a[r]); // 將第二個指針指到的元素新增至 deque 裡面
33
34         if( mp[a[r]] == 0 ) total++;
35
36         // 如果這個元素在 [l,r] 這個區間還沒出現過，則不同的數字數量 +1
37
38         mp[a[r]]++; // a[r] 這個元素在 [l,r] 出現的次數 +1
39
```

當兩個指針圍成的區間內超過 K 種不同的數字時

```
40     if( total > k ) // 如果 [l,r] 內已經超過 k 種不同的數字了
41     {
42         while( total > k ) // 開始移動第一個指針
43         {
44             ans += ( r - 1 ) - l + 1; // 移動第一個指針時順便計算答案
45
46             int num = dq[0]; // 取得目前區間 [l,r] 的第一個元素
47
48             dq.pop_front(); // 將第一個元素刪除
49
50             mp[num]--; // num 在區間內的出現次數 -1
51
52             if( mp[num] == 0 ) total--;
53
54             // 如果 mp[num] = 0 則表示 num 已經沒出現在區間內了，所以不同的數字數量 -1
55
56             l++; // 移動第一個指針
57         }
58     }
```

將 $[l, r]$ 內不同元素的數量處理成 $\leq k$ 時

```
60      r++; // 移動第二個指針
```

小細節：當 $r > n$ 的時候

你會發現，我們更新答案只有在 $[l, r]$ 內的不同元素的數量 $> k$ 時才更新

那麼當 $r = n + 1$ 的時候會少更新一次答案

舉例：

如果 $[4, 8]$ 這個區間是 ok 的，但是這個序列的長度只有 8

那麼當第二個指針到達 9 這個位置時會因為序列長度而離開迴圈

這時我們少將 $[4, 8]$ 這組區間計算到答案當中

記得最後要再更新一次答案

```
63     int u1 = 1 , u2 = ( n - 1 ) - l + 1; // 最後區間內的元素個數會有 ( n - 1 ) - l + 1 個
64
65     // [ l , r ] 這個區間內的所有子區間都是答案
66     // 因此我們利用等差級數和的公式即可以計算 [ l , r ] 內共有幾個子區間
67
68     ans += ( u1 + u2 ) * ( u2 - u1 + 1 ) / 2; // 等差級數和公式
69
70     cout << ans << "\n";
```


這題的 Two Pointers 寫法有很多種

- 用自己理解下來的邏輯去寫看看
- 不一定要照著簡報的範例
 - 簡報的範例 code 可以不使用 deque
 - 用指針來存取元素就可以了