Chapter 4
資料結構
Data Structure

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

- STL (Standard Template Library) 工具的使用
  - vector
  - queue
  - pair
  - deque
  - priority_queue
  - set / multiset / unordered_set
  - map / unordered_map

迭代器 iterator

- 一種類似指標的變數型態
- 儲存的東西是 STL 元素的地址
- 部分的 STL 工具支援迭代器的操作 (queue、stack、priority_queue 不支援)
- 可以讓你在 STL 的使用上做出更多的變化

迭代器 iterator

- 宣告迭代器
  - [STL 工具的類型]<型態>::iterator [name]

```
59      vector<int>::iterator name;
```

迭代器 iterator

- 取值 (取得該迭代器地址上住的人是誰)
- *[迭代器變數]

```
cout << *it;
```

迭代器 iterator 的操作

- 往正方向移動一格
- 語法：[迭代器名稱]++
- 所有類型的迭代器都可以做這一個操作

```
61      it++;
```

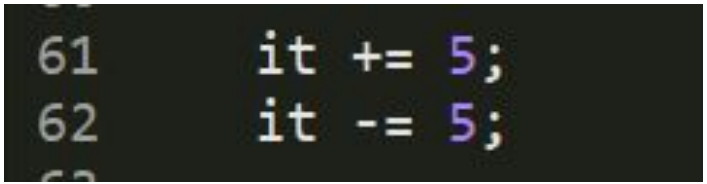
迭代器 iterator 的操作

- 往負方向移動一格
- 語法：[迭代器名稱]--
- 雙向迭代器可以做到這一個操作

```
61      it--;
```

迭代器 iterator 的操作

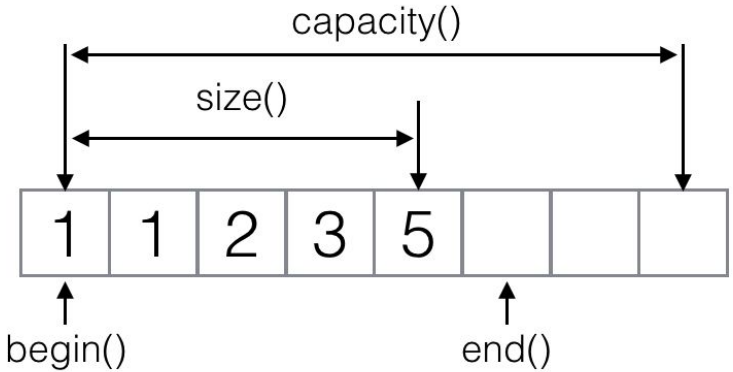
- 往正方向或負方向移動數個位置
- 語法：[迭代器名稱] [+ or -] [移動的數量]
- 隨機迭代器可以做到這一個操作



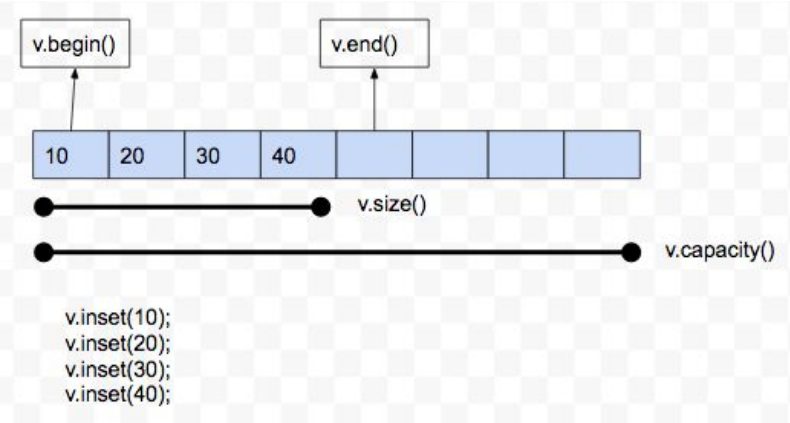
.rbegin() 與 .rend()

rbegin() 是指向最後一個元素的反向迭代器（也就是 .end() - 1 這個位置）

rend() 是指向第一個元素之前的反向迭代器（也就是 .begin() - 1 這個位置）



vector



- #include <vector>
- 宣告 vector <Type> name;
- vector 的迭代器屬於隨機迭代器
- vector 的常見操作
  - .push_back(val); // O(1) (在最右邊新增元素 val)
  - .emplace_back(val); // O(1) (在最右邊新增元素 val)
  - .pop_back(); // O(1) (把最右邊的那一個元素刪除)
  - .size(); // O(1) (取得當前 vector 的長度)
  - .erase(iterator); // O(n) (把該迭代器位置的東西刪除)
  - .clear(); // 把 vector 清空 O(元素個數)
  - vector 可以像一般陣列一樣，隨機存取某一個位置的 value

vector

```
10      vector<int> v;
11
12      v.push_back(1); // [1]
13      v.push_back(2); // [1,2]
14
15      cout << v.size() << "\n"; // 2
16
17      v.pop_back(); // [1]
18
19      cout << v.size() << "\n"; // 1
20      cout << v[0] << "\n"; // 1
21
22      v.erase(v.begin());
23
24      cout << v.size() << "\n"; // 0
```

```
10      vector<int> v;
11
12      v.push_back(3);
13
14      cout << *v.begin(); // 3
```

```
10      vector<int> v;
11
12      v.push_back(3);
13      v.push_back(4);
14
15      cout << *(v.begin()+1); // 4
```

vector

```
10      vector<int> v;
11
12      v.push_back(1); // [1]
13      v.push_back(2); // [1,2]
14
15      cout << v.size() << "\n"; // 2
16
17      v.pop_back(); // [1]
18
19      cout << v.size() << "\n"; // 1
20      cout << v[0] << "\n"; // 1
21
22      v.erase(v.begin());
23
24      cout << v.size() << "\n"; // 0
```

```
10      vector<int> v;
11
12      v.push_back(3);
13
14      cout << *v.begin(); // 3
```

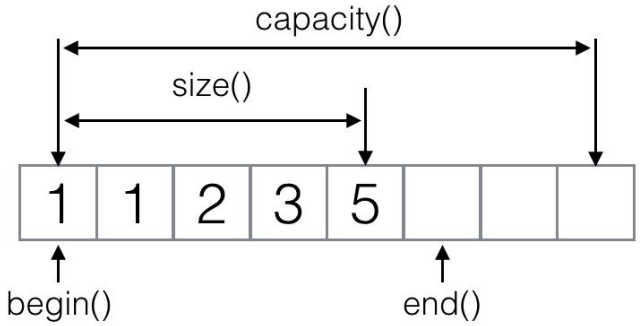
```
10      vector<int> v;
11
12      v.push_back(3);
13      v.push_back(4);
14
15      cout << *(v.begin()+1); // 4
```

vector 其他常見的用法

宣告好大小：vector <Type> your_vector_name(SET_SIZE);

二維陣列的 vector：vector <Type> your_vector_name[SET_SIZE];

初始化：vector <Type> ouob(n,num); // 將一個長度為 n 的 vector 都初始化為 num



二維 vector

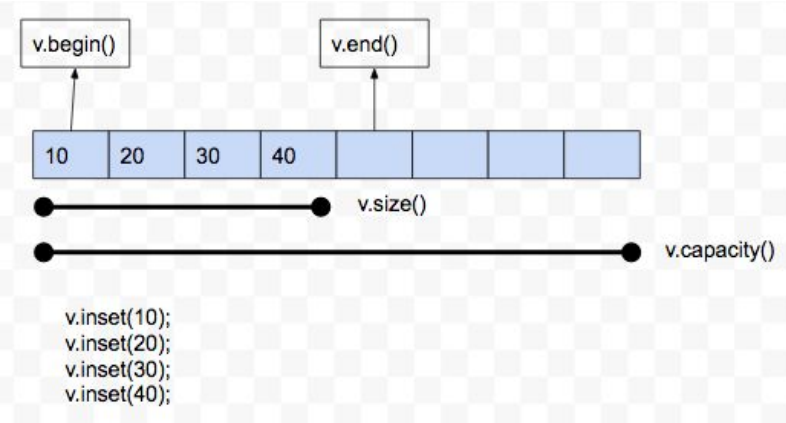
```
 7     vector <int> a[5];
 8
 9     a[4].push_back(1);
10
11     cout << a[4][0] << "\n"; // 1
12
13     a[4].push_back(5);
14
15     cout << a[4][1] << "\n"; // 5
16
17     a[0].push_back(4);
18
19     cout << a[0][0] << "\n"; // 4
```

練習題： CSES Problem Set Static Range Sum Queries

練習使用 vector 來建立前綴和吧！

```
13    int n,q;
14
15    cin >> n >> q;
16
17    vector <int> a(n+1),b(n+1,0);
18
19    for(int i=1;i≤n;i++) cin >> a[i] , b[i] = b[i-1] + a[i];
20
21    while(q--)
22    {
23        int l,r;
24
25        cin >> l >> r;
26
27        cout << b[r] - b[l-1] << "\n";
28    }
```

deque



- #include <deque>
- 宣告 deque <Type> name;
- deque 的迭代器屬於隨機迭代器
- deque 的常見操作
  - vector 有的操作 deque 都有
  - .push_front(val); // 在最前面新增元素 val O(1)
  - .pop_front(); // 把最前面的元素刪掉 O(1)
  - deque 的常數比 vector 大，小心使用，不要太濫用

deque

```
12      deque<int> dq;
13
14      dq.push_back(1); // 1
15      dq.push_back(2); // 1,2
16
17      dq.push_front(3) // 3,1,2
18      dq.pop_front(); // 1,2
```

deque

```
12      deque <int> dq;
13
14      for(int i=1;i≤5;i++) dq.push_back(i); // 1 2 3 4 5
15
16      dq.clear();
17
18      cout << dq.size() << "\n"; // 0
19
20      for(int i=1;i≤5;i++) dq.push_front(i); // 5 4 3 2 1
21
22      cout << dq[0] << "\n"; // 5
23
24      dq.pop_front();
25
26      cout << dq[0] << "\n"; // 4
```

pair

- #include <utility>
- 宣告 pair <Type1,Type2> name;
- pair 的常見操作
  - pair 就是把兩個東西綁在同一個變數上的工具
  - 第一個變數為 .first、第二個變數為 second
  - 把兩個變數綁成 pair 可以使用 {val1,val2} 或 make_pair(val1,val2)

```
12    pair<int,int> a;
13
14    a.first = 1;
15    a.second = 2;
16
17    cout << a.first << " " << a.second << "\n"; // 1 2
18
19    pair<string,int> b;
20
21    b.first = "Hello";
22    b.second = 2;
23
24    cout << b.first << " " << b.second << "\n"; // Hello 2
```

vector with pair

```
12      vector<pair<int,int>> a;
13
14      for(int i=0;i<5;i++) a.push_back({i,i+1});
```

pair sorting

- pair 的排序預設為先比 first 再比 second
- 要自己更改規則的話必須自己寫一個 compare function

```
12      vector<pair<int,int>> a;
13
14      for(int i=0;i<5;i++) a.push_back({i,i+1});
15
16      sort(a.begin(),a.end());
```

stack 堆疊

- LIFO (Last In First Out) 的結構
- 使用前：#include <stack>
- 宣告：stack <Type> name;
- .push(val) 新增一個元素到頂端 O(1)
- .top(); 查看最上面的元素 O(1)
- .pop() 刪除最上面的元素 O(1)
- .size() 取得堆疊長度 O(1)
- stack 不能使用 clear
- stack 是無法使用迭代器的，務必特別留意

```
13    stack <int> s;
14
15    for(int i=0;i≤5;i++) s.push(i); // 0 1 2 3 4 5
16
17    cout << s.top() << "\n"; // 5
18
19    s.pop(); // 0 1 2 3 4
20
21    cout << s.top() << "\n"; // 4
```

當 stack 內沒有元素時

當 stack 內部沒有元素時是無法使用 top() 與 pop() 的

如果在沒有元素時使用這兩個操作程式會直接當掉（Runtime Error）

經典問題：Zerojudge a565 p&q 的邂逅

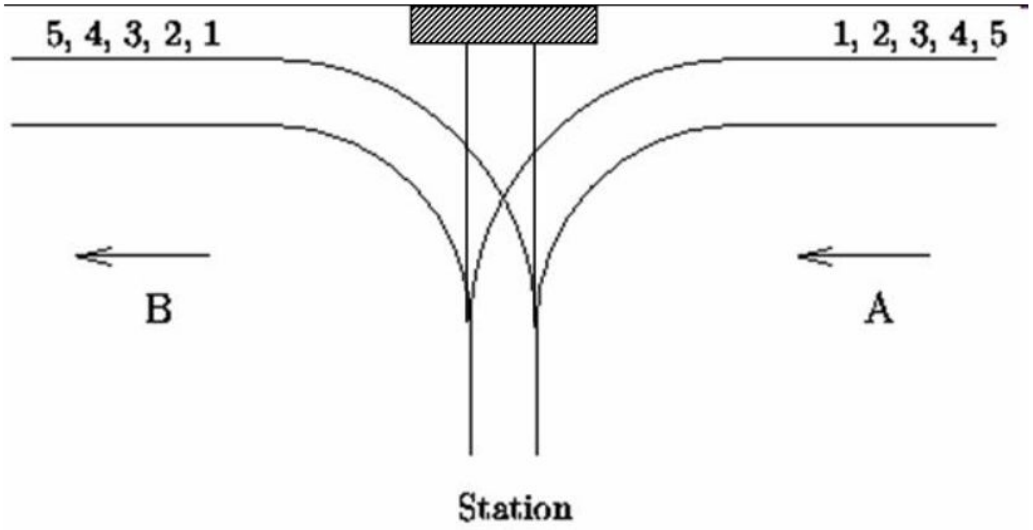
類似括號匹配的題目，需要觀察的是

你會發現，每一個 q 都只會跟距離他最近且還沒有跟其他 q 匹配的 p 做匹配

關鍵程式碼

```
23          string s;
24
25          cin >> s;
26
27          stack <int> st;
28
29          int ans = 0; // 計算答案
30
31          for(int i=0;i<s.size();i++)
32          {
33              if( s[i] == 'p' ) st.push(1); // 發現 p ; 拉近 stack 等待匹配
34
35              else if( s[i] == 'q' )
36              {
37                  if( st.size() != 0 ) // 這一行非常重要，沒有這一行可能會 RE
38                  {
39                      st.pop(); // 將當前離 q 最近且等待匹配的 p 移除掉（匹配成功）
40
41                      ans++; // 答案 + 1
42                  }
43              }
44          }
45
46          cout << ans << "\n";
```

stack 練習題 (有一點小難度)：TIOJ 1012. Rails



queue

- #include <queue>
- 宣告 queue <Type> name;
- queue 沒有迭代器
- queue 的常見操作
  - .push(val); // 把 val 放到隊伍的尾端 O(1)
  - .pop(); // 把隊伍最前面的人刪除 O(1)
  - .front(); // 看看隊伍最前面的那一個人是誰 O(1)
  - .size(); // 取得 queue 的長度 O(1)
  - queue 無法使用 clear

```
12      queue<int> q;
13      q.push(1);
14      cout << q.front() << "\n"; // 1
15      q.push(2);
16      cout << q.front() << "\n"; // 1
17      q.pop();
18      cout << q.front() << "\n"; // 2
```

練習題：Zerojudge e447 queue 練習題

queue 基本操作練習題

set 集合（支援 O(N) clear ）

- 使用前：#include <set>
- 宣告：set <Type> name;
- .insert(val); 插入 val 到集合裡面 O(logN)
- .erase(val or val's interator); 刪除集合內的 val O(logN)
- .count(val); 查詢集合內 val 的個數 O(val個數)
- .size(); 查詢集合的大小 O(1)
- set 屬於雙向迭代器，移動迭代器只能使用 ++ 或 - - 來做移動

- set 是支援迭代器操作的，但是由於 set 的記憶體位置不具有連續性
- 因此無法將兩個 set 的迭代器做相加或相減的動作
- set 集合內的元素是不會重複的 (非多重集)，這一點非常重要！！！
- 如果插入兩個相同的元素的話，在 set 裡面只會出現一個有唷！

set 集合（支援 O(N) clear ）

```
13      set <int> s;
14
15      for(int i=5;i≥1;i--) s.insert(i); // 1 2 3 4 5
16
17      s.insert(1); // 1 2 3 4 5
18
19      cout << s.count(1) << " " << s.size() << "\n"; // 1 5
20
21      cout << *s.begin() << " " << *s.begin() + 2 << "\n"; // 1 3
22
23      s.erase(1); // 2 3 4 5
24
25      cout << *s.begin() << "\n"; // 2
```

set 遍歷示範 複雜度：O(N)

補充：set 內部元素 .begin() 到 .end() 的大小預設為由小到大

```
13      set <int> s;
14
15      for(int i=5;i≥1;i--) s.insert(i); // 1 2 3 4 5
16
17      for(int i=1;i≤6;i++) s.insert(i); // 1 2 3 4 5 6
18
19      for(auto i=s.begin();i≠s.end();i++) // 由於 set 迭代器位置不能比較，因此一定要寫不等於
20      {
21          cout << *i << " "; // 1 2 3 4 5 6
22      }
23
24      cout << "\n";
25
26      for( auto i : s ) cout << i << " "; // 這個寫法做的事情跟上面那一個一樣
27
28      cout << "\n";
```

set 元素獨一無二

```
12    set <int> s;
13
14    for(int i=5;i≥1;i--) s.insert(i);
15
16    for(auto i=s.begin();i≠s.end();i++) cout << *i << " "; // 1 2 3 4 5
17
18    cout << "\n";
19
20    s.insert(1);
21
22    cout << s.count(1) << " " << s.size() << "\n"; // 1 5
23
24    for(auto i=s.begin();i≠s.end();i++) cout << *i << " "; // 1 2 3 4 5
25
26    cout << "\n";
27
28    cout << *s.begin() << "\n"; // 1
29
30    s.erase(1);
31
32    cout << *s.begin() << "\n"; // 2
```

補充：set 的 insert(val) 當中的 first and second

使用 set 做 insert 時，其實 insert 會回傳一個 pair 回來

這個 pair 的第一個參數是一個迭代器的型態，表示這個 val 被插入的迭代器位置

第二個參數是一個布林的型態，表示 val 是否有被插入成功

```
14      set <int> s;
15
16      cout << s.size() << "\n"; // 0
17
18      pair <set<int>::iterator,bool> u = s.insert(1);
19
20      cout << *u.first << " " << u.second << "\n"; // 1 1(true)
21
22      u = s.insert(1); // 已經有 1 了，插入失敗
23
24      cout << u.second << "\n"; // 0(false)
```

補充：.find(val); // set::find or map::find

使用前：#include <algorithm>

時間複雜度：O(logN)

```
13    set <int> s;
14
15    for(int i=5;i≥1;i--) s.insert(i); // 1 2 3 4 5
16
17    for(int i=1;i≤6;i++) s.insert(i); // 1 2 3 4 5 6
18
19    auto it = s.find(6);
20
21    cout << *it << "\n"; // 6
```

回傳 (迭代器型態)：回傳 val 在這個 STL 的迭代器位置

如果找不到 val 則回傳 .end()

find 也可以指定區間來做尋找，不過因為不常用因此這邊沒有舉例出來

練習題：Ten Point Round #3 (Div. 2) pH 獨一無二的完美數列

注意 set 元素獨一無二的特性

```
13      set <string> s;
14
15      for(int i=1;i≤100;i++) s.insert("ouob");
16
17      cout << *s.begin() << " " << s.size() << "\n"; // ouob 1
```

set & find：APCS 2021 1 月 第三題 切割費用

- find 與迭代器的互相配合

set 練習題：Ten Point Round #8 藤原千花與字串

- Hint: 字典序與 set 由小到大的特性

multiset

操作上與特性基本上同 set

(標頭檔跟 set 一樣)

```
13    multiset <int> s;
14
15    for(int i=1;i≤100;i++) s.insert(1);
16
17    cout << s.size() << " " << s.count(1) << "\n"; // 100 100
```

唯一不同的是：

multiset 可以放入許多相同的元素

需要特別注意的是在 count 的時候複雜度是 O(查詢的元素個數 + log N)

宣告：multiset <Type> name;

multiset 的 erase

特別注意 erase 如果放的是 val 的值

那麼就算是 multiset 也會把集合內的 val 全部刪除

如果只想刪在 multiset 裡面的其中一個 val 元素，那麼必須先使用 find 隨機找到一個 val 的迭代器位置之後再使用 erase 將該迭代器位置刪除

```
13      multiset <int> s;
14
15      for(int i=1;i≤100;i++) s.insert(1);
16
17      cout ≪ s.size() ≪ " " ≪ s.count(1) ≪ "\n"; // 100 100
18
19      s.erase(s.find(1)); // 只刪 1 個
20
21      cout ≪ s.size() ≪ " " ≪ s.count(1) ≪ "\n"; // 99 99
22
23      s.erase(1); // 刪全部
24
25      cout ≪ s.size() ≪ " " ≪ s.count(1) ≪ "\n"; // 0 0
```

set 取得第一個元素與最後一個元素

雖然 set 只能使迭代器 ++ -- ， 不過我們也可以利用 rbegin() 這個反向迭代器取得最後一個元素是誰

set 裡的最後一個元素也就會是最大的元素（ set 由小到大依序遞增的特性 ）

```
12    set <int> s;
13
14    for(int i=1;i≤5;i++) s.insert(i);
15
16    cout << *s.begin() << " " << *s.rbegin() << "\n"; // 1 5
17
18    auto it = s.end();
19
20    it--;
21
22    cout << *it << "\n"; // *s.rbegin() 5
```

map 映射（支援 O(N) clear ）

```
13    map <int,int> mp;
14
15    mp[-1] = 5;
16
17    map <string,string> mp2;
18
19    mp["ouob"] = "Yez orz";
20
21    cout << mp[-1] << " " << mp2["ouob"] << "\n";
```

- 使用前：#include <map>
- 宣告：map <Type1,Type2> name;
- Type1 是 index 的型態，Type2 是 val 的型態
- 舉例：
- map <string,int> mp;
- mp[“ouob”] = 5; /* O(logN) */
- map 也屬於雙向迭代器，移動時只能使用 ++ 或是 - - 來移動

- map 跟 set 一樣因為迭代器位置不具有連續性，因此無法使用兩個迭代器做算術運算

map

看到 map 的操作之後你會發現

map 其實就跟平常在使用陣列一樣，只是 map 的索引值將不再有限制

可以是字串，也可以是負數，甚至是浮點數等等

在 map 的世界裡，這些東西都可以拿來當索引值！

小細節：map 的 size 操作（回傳有幾個索引被指定）

```
12      map <int,string> mp;
13
14      mp[5] = "Yez"; // 5 這個索引被指定
15
16      mp[1] = "ttt"; // 1 這個索引被指定
17
18      cout << mp.size() << "\n"; // 2
```

如果 map 的該索引沒有被指定，會是什麼

map 的索引如果是沒被指定的狀態，預設的值如下

- 整數 = 0
- 字元 = '' // 空字元
- 字串 = "" // 空字串

但這個其實是一個不太正規的檢查方法，可能會因為編譯器有所變化

如果你想要檢查 map 的該索引有沒有被指定，請使用 map.count(index)

map 的 count O(logN)

用法：.count(index);

回傳：如果 index 有被指定值則回傳 1，沒被指定值回傳 0

```
12      map <string,int> mp;
13
14      mp["Yez"] = 1;
15
16      cout << mp.count("Yez") << "\n"; // 1
17
18      cout << mp.count("EEEEE") << "\n"; // 0
```

map 遍歷

補充：map 內部元素 .begin() 到 .end() 的大小預設為由小到大

且 map 預設每一個位置為 pair 型態，.first 為索引值，.second 為該索引位置的值

```
13    map <int,int> mp; // map 本身就是一個 pair，索引值是 first , 值是 second
14
15    for(int i=1;i≤5;i++) mp[i] = i + 1;
16
17    for(auto i=mp.begin();i≠mp.end();i++) // 由於迭代器不能比大小，因此要寫 ≠
18    {
19        cout << (*i).first << " " << (*i).second << "\n"; // 一定要連 * 都小括號起來不然會衝突
20
21        // 1 2
22        // 2 3
23        // 3 4
24        // 4 5
25        // 5 6
26    }
27
28    for( auto i : mp ) cout << i.first << " " << i.second << "\n"; // 同上輸出
```

map 延伸：CSES Problem Set Subarray Sums I

前綴和 與 map 互相搭配用來維護狀態的題目

備住：這題是一道經典題

前綴和

一個區間 [L , R] 的和是由兩個前綴和序列的項相減而來的

因此我們枚舉 R，假設我們當前位於 4 這個位置

就代表我們必須找到一個 L，使得 [L , 4] 這個區間的前綴和會等於 x

如果 [L , 4] 的和等於 x 就代表

前綴和陣列的第 4 項 跟 前綴和陣列的第 L - 1 項相減會等於 x

前綴和

因此我們利用 map 紀錄所有當前的前綴和值數量

並在枚舉每一個 R 的時候檢查有幾個剛好等於 當前總和 - x 的前綴和

並將這個數量加上去就可以求出最後的數量了

這是一個非常經典的技巧，務必要學起來！

輸入處理

```
13      int n,m;
14
15      cin >> n >> m;
16
17      vector <int> a(n);
18
19      for(int i=0;i<n;i++)
20      {
21          cin >> a[i];
22      }
23
24      map <int,int> mp;
```

計算答案

```
24      map <int,int> mp;
25
26      int ans = 0,total = 0; // total : 當前的前綴和
27
28      for(int i=0;i<n;i++)
29      {
30          total += a[i];
31
32          if( total == m ) ans++; // 如果 [0,i] 的和為答案，則答案數量 +1
33
34          if( mp[total-m] != 0 ) // 如果要找到一組區間 [L,i] 的和為 m
35          {       // 則表示要找到至少一個位置的前綴和為 total - m 才會使 total - ( total - m ) = m
36              ans += mp[total-m]; // 如果有 total - m 的前綴和，則將答案加上其數量
37          }
38
39          mp[total]++; // total 這個前綴和的數量 +1
40      }
41
42      cout << ans << "\n";
```

priority_queue 優先佇列

- 基本上大部分同 queue 的操作與性值（標頭檔跟 queue 一樣）
- 宣告：priority_queue <Type> name;
- 不同的是，priority_queue 預設先拿出來的東西為最大值
- 且取得當前最大的元素的操作為 .top(); 而非 .front();

- priority_queue 每次 push 與 pop 的時間複雜度皆為 O(logN)、top() 為 O(1)
- 補充：如果想要改成最小值宣告可改成：
- priority_queue <Type,vector<Type>,greater<Type>> name;

priority_queue

```
13    priority_queue <int> pq1; // 以大優先的 pq
14
15    priority_queue <int,vector<int>,greater<int>> pq2; // 以小優先的 pq
16
17    for(int i=1;i≤5;i++) // 1 2 3 4 5
18    {
19        pq1.push(i);
20
21        pq2.push(i);
22    }
23
24    cout << pq1.top() << " " << pq2.top() << "\n"; // 5 1
25
26    pq1.pop();
27
28    pq2.pop();
29
30    cout << pq1.top() << " " << pq2.top() << "\n"; // 4 2
```

練習題：Ten Point Round #10 pE 最高價值的交換

priority_queue 用法練習題

pq 進階應用：Ten Point Round #7 (Div. 1) pC 中位數

priority_queue 的進階應用，這個技巧是一個經典技巧

最暴力的作法

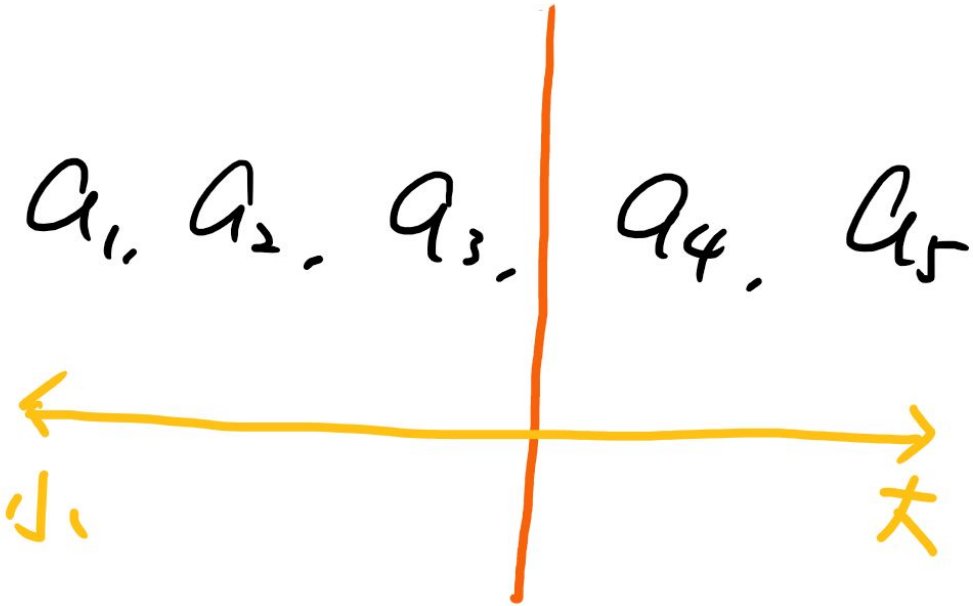
最直觀的方法：每次新增元素後就 sort 一次

但是這題最多可能會新增 10 的 6 次方個數字

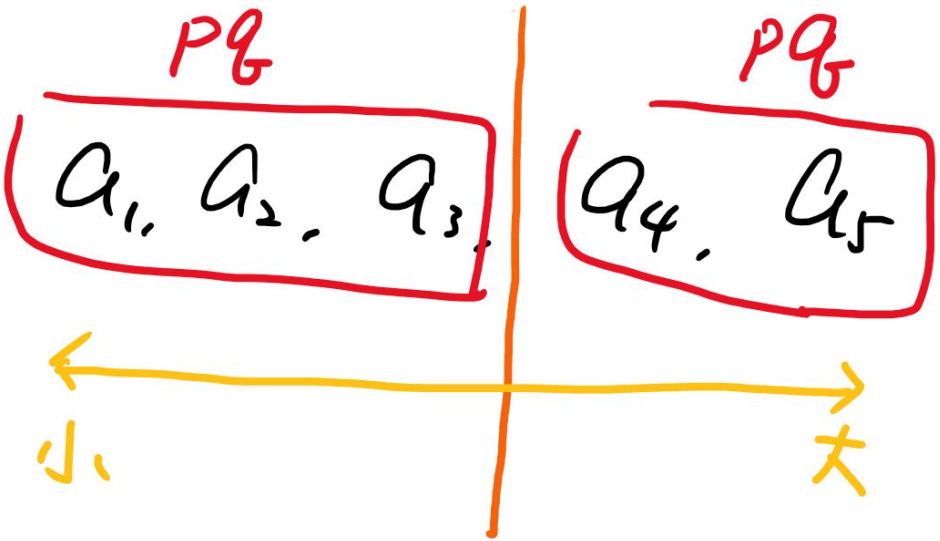
sort 10 的 6 次方次時間複雜度很明顯是不夠好的

那我們究竟應該要怎麼用 priority_queue 維護我們的中位數呢？

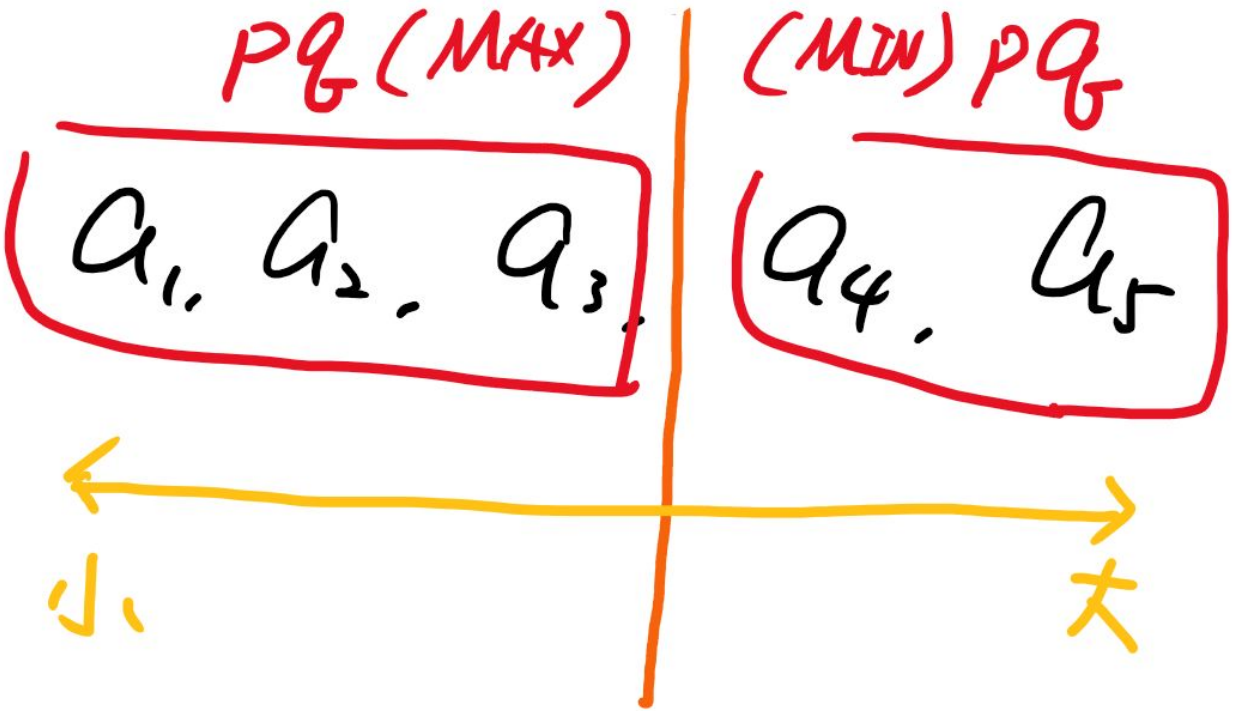
怎麼維護？我們如果將序列切成一半



左右各開一個 priority_queue 來維護

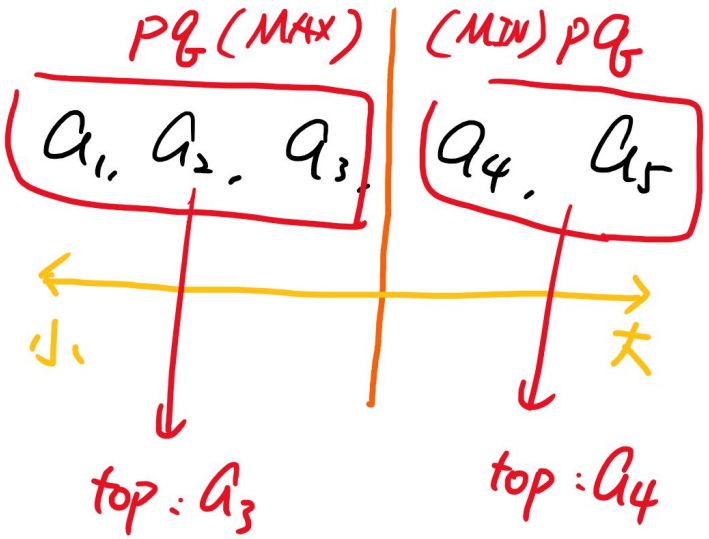


左 pq 以大優先、右 pq 以小為優先



這時你會發現，如果我們這樣子去維護

我們只需要拿兩個 priority_queue 的 top 就可以直接算中位數了！



因此在每次新增新的數字時，我們要維護好兩邊的 pq

首先，要既然是中位數，那麼我們一定要維護好兩邊 pq 的長度

不然一長一短所取得的值肯定不會是中位數

但是我們在插入元素的時候會有一些情況

如果插入的元素大小 < 左邊的 pq 的 top() 那麼這次插入一定插入在左邊

其餘的情況則都插入在右邊

插入完左邊或右邊之後

別忘記插入完要維護好兩邊的平衡，不然取得的中位數會是錯的

實作看看吧！

```
7 priority_queue <int,vector<int>,greater<int>> r; // 維護右邊
8
9 priority_queue <int> l; // 維護左邊
```

插入元素後維護長度

```
19      while( cin >> n )
20      {
21          if( n == 0 ) break;
22
23          if( l.size() == 0 || l.top() > n ) // 如果 l 的 top 比 n 還大，那 n 一定只能放左邊
24          {
25              l.push(n);
26          }
27          else
28          {
29              r.push(n); // 不能放左邊則放右邊
30          }
31
32          if( l.size() > r.size() + 1 ) // 如果左邊已經多出了 2 個元素，從右邊抓一個過來
33          {
34              r.push(l.top());
35
36              l.pop();
37          }
38          else if( l.size() + 1 < r.size() ) // 如果右邊已經多出了 2 個元素，從左邊抓一個過來
39          {
40              l.push(r.top());
41
42              r.pop();
43          }
```

計算答案

```
45          if( l.size() == r.size() ) // 這時的序列長度為偶數
46          {
47              cout << ( r.top() + l.top() ) / 2 << "\n";
48          }
49          else if( l.size() > r.size() ) // 左邊多一個元素、中位數是左邊的 top
50          {
51              cout << l.top() << "\n";
52          }
53          else // 中位數是右邊的 top
54          {
55              cout << r.top() << "\n";
56          }
```

CSES: Sliding Median

You are given an array of n integers. Your task is to calculate the median of each window of k elements, from left to right.

The median is the middle element when the elements are sorted. If the number of elements is even, there are two possible medians and we assume that the median is the smaller of them.

Input

The first input line contains two integers n and k: the number of elements and the size of the window.

Then there are n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print $n-k+1$ values: the medians.

Constraints

- $1 \le k \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:
```
8 3
2 4 3 5 8 1 2 1
```

Output:
```
3 4 5 5 2 1
```

CSES: Sliding Median

- 其實就把他想像成是一個動態的中位數，支援插入元素刪除元素
- 但是 priority_queue 不支援刪除，怎麼做？
- 想想看，有沒有一個資料結構，可以動態找最大，最小 且支援刪除

CSES: Sliding Median

- 其實就把他想像成是一個動態的中位數，支援插入元素刪除元素
- 但是 priority_queue 不支援刪除，怎麼做？
- 想想看，有沒有一個資料結構，可以動態找最大，最小 且支援刪除
- set！
- 所以你只要將 priority_queue 改寫成 set 就可以解決這一題了
- 只是要記得使用 multiset，因為元素會有所重複
- 這題留給大家練習看看

STL 的資訊量好大

STL 有著非常多的細節與使用的方法需要記，因此一開始一定會很難適應

平時可以善加利用 cppreference 來查詢相關的資料或者是用法

另外，大量的練習也是熟悉基本 STL 很重要的一點哦！