

STL 與基礎資料結構

sam571128

September 23, 2021

課前公告

我們的社團有個 discord 的群組

大家到 Codeforces 的 Group 的話會有連結可以加入

然後上禮拜的題目有的題目，由於測資出錯，有進行調整

今天要教的東西

今天要教的東西可能會偏語法，要講關於 STL 的部分
已經熟悉語法的社員們可能要等到課程後半才會有較難的東西

STL?

STL 標準模板庫 (Standard Template Library)

- C++ 內建的模板庫
- 有許多寫好的函數與資料結構可以使用
- 多數比自己手寫的快

STL 中的函數

最常用的幾個函數

- `sort(陣列開始, 陣列結尾)` 排序 $O(n \log n)$
- `memset(陣列, 數值, sizeof (陣列))` 賦值給陣列 $O(n)$
- `lower_bound(陣列開始, 陣列結尾, x)` 找第一個 $\geq x$ 的值 $O(\log n)$

sort() 用法

```
sort(arr, arr+n);
```

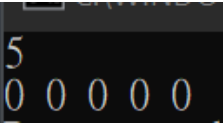
5				
5	2	3	4	1
1	2	3	4	5

使用之後，會將陣列排序

他的時間複雜度會是 $O(n \log n)$ ，因此不用額外手寫排序

memset() 用法

```
int arr[n];  
memset(arr, 0, sizeof(arr));
```



使用之後，會將陣列初始化為你要的值 (但是是位元)

他的時間複雜度會是 $O(\text{陣列長度})$ ，不過能初始化的數值有限

一般只拿來使用在初始化 0, -1, ∞ (0x3f3f3f3f)

建議使用 fill(陣列開始, 陣列結尾, x) 取代

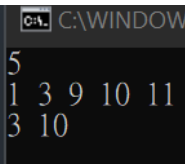
fill() 用法

```
int arr[n];  
fill(arr, arr+n, 0);
```

使用之後，會將陣列初始化為你要的值

lower_bound() 用法

```
int* pos = lower_bound(arr, arr+n, 10);  
int idx = pos-arr;  
  
cout << idx << " " << arr[idx] << "\n";
```



C:\WINDOW
5
1 3 9 10 11
3 10

(僅能使用在排序後的陣列!) 使用後可以找到第一個大於 x 的位置

今天要教的東西

稍微偏題了，讓我們會到今天的主軸
我們今天要教的有以下幾種資料結構

- vector, deque
- stack, queue
- set, map, priority_queue (heap)
- dsu (disjoint set union)

動態陣列 - vector

是否總是覺得陣列很麻煩，一定要固定大小呢？

如果有個東西可以讓我們隨時改變他的大小是不是方便很多呢？

而可以做到這點的就是動態陣列，在不同程式語言有不同的名稱

但在 C++ 就叫做 vector，也就是向量

動態陣列 - vector

```
vector<int> v;
```

- 宣告方式: vector< 資料型態 > 變數名稱
- 加入元素到後方: push_back(x) (x 為要加入的元素)
- 移除最後一個元素: pop_back()
- 查詢大小: size()
- 是否為空: empty()
- 改變大小: resize(大小)
- 清除元素: clear()

動態陣列 - vector

他基本上是我們最常拿來使用的一種 STL

有人甚至直接拿它取代陣列使用

只要陣列的大小不是固定的都可以拿來使用

動態陣列 - vector

不過為了讓大家先熟悉使用這個資料結構

我準備了一題可以發揮 vector 效用的題目給大家練習

在這裡先讓大家練習

動態陣列 - deque

```
deque<int> arr;
```

- 念法: deck 或 de-que
- 另稱: 雙向隊列
- 宣告方式: deque< 資料型態 > 變數名稱
- 加入元素到前方: push_front(x) (x 為加入的元素)
- 移除最前面的元素: pop_front()
- 其餘用法與 vector 完全相同

動態陣列 - deque

使用時機:

- 滑動窗口 (Sliding Window): 固定區間長度
- 雙指針 (Two Pointers): 下週會提

動態陣列 - deque

Sliding Sum

給你一個 n ($1 \leq n \leq 3 \times 10^6$) 項的陣列，以及一個數字 k ($1 \leq k \leq n$)，詢問每個長度為 k 的子區間中的總和為？

範例：若陣列為 $[5, 4, 2, 1, 3]$ ，輸出 $11, 7, 6$ ($[5, 4, 2], [4, 2, 1], [2, 1, 3]$)

動態陣列 - deque

Sliding Sum

給你一個 n ($1 \leq n \leq 3 \times 10^6$) 項的陣列，以及一個數字 k ($1 \leq k \leq n$)，詢問每個長度為 k 的子區間中的總和為？

範例：若陣列為 $[5, 4, 2, 1, 3]$ ，輸出 11, 7, 6 ($[5, 4, 2]$, $[4, 2, 1]$, $[2, 1, 3]$)

遇到這種題目，要怎麼處理呢？

$n \leq 3 \times 10^6$ ，如果每一次都從某個點開始找 k 個數字

以時間複雜度 $O(nk)$ 估計會 TLE

那我們要怎麼做呢

動態陣列 - deque

Sliding Sum

給你一個 n ($1 \leq n \leq 3 \times 10^6$) 項的陣列，以及一個數字 k ($1 \leq k \leq n$)，詢問每個長度為 k 的子區間中的總和為？

範例：若陣列為 $[5, 4, 2, 1, 3]$ ，輸出 $11, 7, 6$ ($[5, 4, 2]$, $[4, 2, 1]$, $[2, 1, 3]$)

注意到每一塊長度為 k 的連續子區間都會有重複的數字

我們可以用類似一個窗口滑過去的方式計算

動態陣列 - deque

Sliding Sum

給你一個 n ($1 \leq n \leq 3 \times 10^6$) 項的陣列，以及一個數字 k ($1 \leq k \leq n$)，詢問每個長度為 k 的子區間中的總和為？

範例：若陣列為 $[5, 4, 2, 1, 3]$ ，輸出 $11, 7, 6$ ($[5, 4, 2]$, $[4, 2, 1]$, $[2, 1, 3]$)

注意到每一塊長度為 k 的連續子區間都會有重複的數字

我們可以用類似一個窗口滑過去的方式計算

動態陣列 - deque

5 4 2 1 3

總和為 $5 + 4 + 2 = 11$

動態陣列 - deque

5 4 2 1 3



總和為 $4 + 2 + 1 = 7$

動態陣列 - deque

5 4 2 1 3

總和為 $2 + 1 + 3 = 6$

動態陣列 - deque

每次都把最前面和最後面的數字加入和移除

由於 deque 的數字加入與移除皆為 $O(1)$

整體時間複雜度為 $O(n)$

動態陣列 - deque

Sliding Minimum

給你一個 n ($1 \leq n \leq 3 \times 10^6$) 項的陣列，以及一個數字 k ($1 \leq k \leq n$)，詢問每個長度為 k 的子區間中的最小值為？

範例：若陣列為 $[5, 4, 2, 1, 3]$ ，輸出 $2, 1, 1$ ($[5, 4, 2]$, $[4, 2, 1]$, $[2, 1, 3]$)

動態陣列 - deque

Sliding Minimum

給你一個 n ($1 \leq n \leq 3 \times 10^6$) 項的陣列，以及一個數字 k ($1 \leq k \leq n$)，詢問每個長度為 k 的子區間中的最小值為？

範例：若陣列為 $[5, 4, 2, 1, 3]$ ，輸出 $2, 1, 1$ ($[5, 4, 2]$, $[4, 2, 1]$, $[2, 1, 3]$)

跟剛剛的題目長得差不多，但是改成找最小值，要怎麼做呢？我們同樣用剛剛的方式去做，會得到什麼結果呢？

動態陣列 - deque

5 4 2 1 3

最小值: $\min(5, 4, 2) = 2$

動態陣列 - deque

5 4 2 1 3



最小值: $\min(4, 2, 1) = 1$

動態陣列 - deque

5 4 2 1 3

最小值: $\min(2, 1, 3) = 1$

動態陣列 - deque

每次都把最前面和最後面的數字加入和移除

由於 deque 的數字加入與移除皆為 $O(1)$

整體時間複雜度為 $O(n)$ (嗎?)

動態陣列 - deque

不過要找數字間的最小值，就不能像總和一樣直接加值或減值了
那怎麼辦呢？

動態陣列 - deque

我們引入一種我們稱為「**單調隊列**」的東西

在 deque 裡存的數字一定是單調的

當我們加入一個比前面的數字小的數字時，我們可以直接移除前面的數字

整個區間的最小值就會在 deque 的最前面

動態陣列 - deque

我們引入一種我們稱為「**單調隊列**」的東西

在 deque 裡存的數字一定是單調的

當我們加入一個比前面的數字小的數字時，我們可以直接移除前面的數字

整個區間的最小值就會在 deque 的最前面時間複雜度: $O(n)$

字串 - String

昨天忘記提到一個很重要的東西了，也就是「字串」

我們昨天講到了 `vector` 這個資料結構

`vector<char>` 是一個很特別的東西，被 C++ 獨立拿出來
變成了另外一種資料結構 `string`

字串 - String

功能與 `vector` 完全一樣，但是多了幾種特殊的操作

像字串 + 字串，字串 + `char` 等等

基本上在打競賽時不會使用 `char[]`

堆疊 - stack

Stack

- 又被稱作堆疊
- 「First in Last Out」
- 想像一疊書疊在桌上，你一定要從最上面拿才可以
- Stack 只能知道最上方的元素是誰，和拿走最上面的元素

堆疊 - stack

```
stack<int> st;
```

- 宣告方式: stack< 資料型態 > 變數名稱
- 加入元素到上方: push(x) (x 為要加入的元素)
- 移除最上方的元素: pop()
- 找最上方的元素: top()
- 查詢大小: size()
- 是否為空: empty()
- 清除元素: **沒有** clear()
- 不可**隨機存取**元素

堆疊 - stack

不過這些功能大家應該覺得 vector 就已經都有了，功能還更強

那為什麼我們要用 stack 呢? (其實不用)

但是 stack 是一種概念

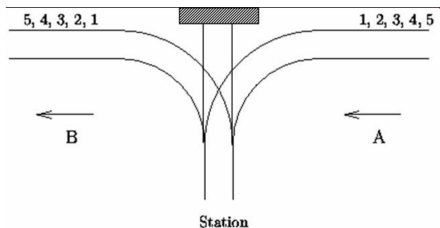
不管你是用陣列、vector 之類的實作都不會影響他的概念

堆疊 - stack

接下來來看一題學過 stack 的人一定會記得的一種題目
也就是「**Rails**」(Zerojudge c123)!

堆疊 - stack

在一個叫「堆疊市」的城市中有一個著名的火車站。由於地形限制以及經費關係，火車站及唯一的鐵路的樣子如下圖：



現在火車從A方向來，預定從B方向離開。火車共有N節車廂，並且各車廂依次以1到N來編號。你可以假設各車廂在進站之前可以單獨與其他車廂分離，也可以單獨離開車站到往B方向的鐵軌或是車站北方的「維修鐵路」上。維修鐵路是一小段至多只能容納M節車廂的鐵軌，可以從車站依照順序將車廂移至維修鐵路，或者將車廂從維修鐵路（如果有的話）駛進車站，但是在把車廂從A開進車站的時候，維修鐵路不能有任何車廂。你可以假設在任何時間火車站都可以容納所有的車廂。但是一旦一節車廂進站後，就不能再回到A方向的鐵軌上了，並且一旦離開車站往B方向後，也不能再回到車站。

現在你的任務是寫一個程式，判斷火車能否以一定的排列方式在B方向的鐵軌上。

堆疊 - stack

這題基本上是一題 stack 最經典的問題之一

簡單來說，有三個車站 A, B, C

在 A 車站中，火車是依照編號排序好的

C 編號車站是一個中繼站，可以將 A 的火車停到 C 上面

然後 B 車站是終點站，題目想問你是否能夠在 B 車站排成某個排列

堆疊 - stack

既然這樣，那我們就分別在 A, C 車站開一個 stack
然後先依照火車數量，將火車依序排進 A 車站的 stack

例如: **頂** 1, 2, 3, 4, 5, 6, 7 **底**

(會發現是依照順序的，其實可以不用開 stack)

接著，我們要在 B 車站排成我們要的排列

例如: **底** 4, 5, 3, 7, 6, 2, 1 **頂**

堆疊 - stack

要做到這一點，假設要先推進 B 的編號為 x

那麼，我們要先將 $1, 2, 3, \dots, x$ 全部都先推到 C 車站上

然後當 C 的頂部為要推進 B 的數字時，就推進去

這樣就可以判斷滿不滿足了

堆疊 - stack

Stack 另外一個很有用的地方是 RBS (Regular Bracket Sequence)

也就是判斷一個括號序列是否是合法的

合法的括號匹配如 `()()`, `((()))()` 等

不合法的括號匹配如 `)(`, `(())`, `((()` 等

堆疊 - stack

我們可以依序輸入 (和)

當輸入 (的時候，就直接推到 stack 上方

當輸入) 的時候，判斷 stack 是否為空，如果有 (，就消除

否則這個括號匹配不合法

當輸入完整個括號匹配之後，判斷 stack 是否為空，如果 stack 不為空，則括號匹配不合法

堆疊 - stack

另外一個常見的地方是包含括號和先乘除後加減的四則運算

假設給你一個算式，如: $1 + (2 \times 3 + 4)$ ，那麼要怎麼寫程式計算呢
(不要用 python 的 `eval()`!)

堆疊 - stack

同時有括號和先乘除後加減有點困難，讓我們先考慮簡單一點的
如果今天什麼都沒有呢？

堆疊 - stack

很簡單吧! 直接從左到右根據運算符號去計算答案就好了

不過如果今天有乘除運算呢?

我們就要去考慮運算的優先順序了

堆疊 - stack

這裡我們引入一種逆波蘭表示法 (Reverse Polish Notation)

(我們平常書寫的東西為中序表示法)

(逆波蘭表示法則是後序表示法)

假設今天的算式長這樣 $(5 + 6) * 7$

根據波蘭表示法，我們寫成這樣 $5\ 6\ +\ 7\ *$

表示我們先計算 $5\ 6\ +$ ，也就是 $5 + 6 = 11$

之後會變成 $11\ 7\ *$ ，也就是 $11 * 7 = 77$

堆疊 - stack

如果今天是 $5 + 6 * 7$

則逆波蘭表示法為 $5\ 6\ 7\ *\ +$ 也就是先進行 $6 * 7$

之後再去執行 $5\ 13\ +$ ，也就是 $5 + 13$

這樣的形式便於我們使用 stack 進行運算

堆疊 - stack

要計算四則運算，其實很簡單，作法就是開兩個 stack
一個維護所有的運算子，另外一個維護所有的數字
程式碼

堆疊 - stack

除了以上幾種應用之外，如同我們昨天所教的「單調隊列」

Stack 也有所謂的「單調 Stack」

如以下例題:

堆疊 - stack

格鬥大賽 (2017 北市賽 p3)

給你 n 個人，每個人有攻擊指數和防禦指數，如果 A 的攻擊指數和防禦指數其中一個大於 B ，另外一個數值不小於 B ，則 A 能夠打贏 B ，問能贏最多場的人是誰

堆疊 - stack

格鬥大賽 (2017 北市賽 p3)

給你 n 個人，每個人有攻擊指數和防禦指數，如果 A 的攻擊指數和防禦指數其中一個大於 B ，另外一個數值不小於 B ，則 A 能夠打贏 B ，問能贏最多場的人是誰

這題也是利用了 Stack 的性質，從左邊和右邊分別開始進行

如果有一個人打贏了 x 個人，那麼打贏那個人的人一定能打贏 $x + 1$ 個人

因為這個緣故，這題也能用 stack 維護

隊列 - Queue

Queue

- 「Last in Last Out」
- 想像一群人排隊，你一定要從最後面進入，最前面離開
- Stack 只能知道最上方的元素是誰，和拿走最上面的元素

隊列 - Queue

```
queue<int> q;
```

- 宣告方式: queue< 資料型態 > 變數名稱
- 加入元素到後方: push(x) (x 為要加入的元素)
- 把前方的元素移除: pop()
- 找最前面的元素: front()
- 查詢大小: size()
- 是否為空: empty()
- 清除元素: **沒有** clear()
- 不可**隨機存取**元素

隊列 - Queue

這個資料結構我們一般會拿來做之後會講的「BFS」

然後他也能拿來做「單調隊列」

不過這裡沒有什麼特別關於 Queue 要提的東西

線性與非線性資料結構

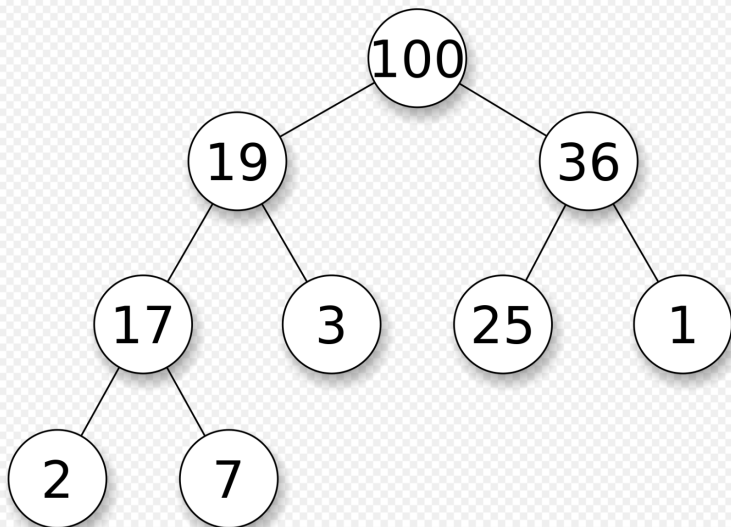
線性

- vector, deque
- stack, queue

非線性 (元素經過排序)

- Heap (Priority Queue)
- Set, Map

堆 - Heap



堆 - Heap

Heap 是一種特殊的資料結構，他是一棵樹

他最上面的節點會是最大/最小值

存最大值的 Heap 稱為 Max Heap

存最小值的 Heap 稱為 Min Heap

堆 - Heap

可以在 $O(\log n)$ 時間加入和移除元素

在 C++ 內建的 Heap 為 `priority_queue<>`

常用在「貪心」的題目

優先隊列 - Priority Queue

```
priority_queue<int> pq;
```

- 宣告方式: priority_queue< 資料型態 > 變數名稱
- 加入元素到後方: push(x) (x 為要加入的元素)
- 把前方的元素移除: pop()
- 找最前面的元素: top()
- 查詢大小: size()
- 是否為空: empty()
- 清除元素: **沒有** clear()
- 不可**隨機存取**元素

優先隊列 - Priority Queue

Min Heap 的宣告如下:

```
prioirty_queue<int,vector<int>,greater<>> pq;
```

優先隊列 - Priority Queue

經典問題

在黑板上，有 n 個數字，每次你可以擦掉兩個數字 a 和 b ，並得到 $a + b$ 分，然後在黑板上寫下 $a + b$ 。一直持續到黑板上只剩一個數字，問你最少會得到多少分

優先隊列 - Priority Queue

經典問題

在黑板上，有 n 個數字，每次你可以擦掉兩個數字 a 和 b ，並得到 $a + b$ 分，然後在黑板上寫下 $a + b$ 。一直持續到黑板上只剩一個數字，問你最少會得到多少分

這個問題其實是我們之後會講到的「貪心」的例題，不過由於他的結論很好猜，所以我們可以用他來理解優先隊列

優先隊列 - Priority Queue

經典問題

在黑板上，有 n 個數字，每次你可以擦掉兩個數字 a 和 b ，並得到 $a + b$ 分，然後在黑板上寫下 $a + b$ 。一直持續到黑板上只剩一個數字，問你最少會得到多少分

很明顯要得到最少分，那就使我們每次拿到的分數最少就好，因此我們只要開一個 Min Heap。每次從中拿出兩個最小的數字，並合併他們丟會去 Min Heap 中。等到 Min Heap 中只剩下一個元素時，總共的分數即為最小分數

優先隊列 - Priority Queue

經典問題

在黑板上，有 n 個數字，每次你可以擦掉兩個數字 a 和 b ，並得到 $a + b$ 分，然後在黑板上寫下 $a + b$ 。一直持續到黑板上只剩一個數字，問你最少會得到多少分

很明顯要得到最少分，那就使我們每次拿到的分數最少就好
因此我們只要開一個 Min Heap。每次從中拿出兩個最小的數字，並合併他們丟會去 Min Heap 中。等到 Min Heap 中只剩下一個元素時，總共的分數即為最小分數

優先隊列 - Priority Queue

經典問題

在黑板上，有 n 個數字，每次你可以擦掉兩個數字 a 和 b ，並得到 $a + b$ 分，然後在黑板上寫下 $a + b$ 。一直持續到黑板上只剩一個數字，問你最少會得到多少分

時間複雜度: $O(n \log n)$

優先隊列 - Priority Queue

```
prioirty_queue<int,vector<int>,greater<>> pq;

for(int i = 0;i < n;i++){
    int x;
    cin >> x;
    pq.push(x);
}

int ans = 0;

while(pq.size() >= 2){
    int a = pq.top(); pq.pop();
    int b = pq.top(); pq.pop();
    ans += a + b;
    pq.push(a+b);
}

cout << ans << "\n";
```

優先隊列 - Priority Queue

Codeforces 1526C2 - Potions (Hard Version)

一開始你的生命值為 0，由左至右有 n 個藥水。你必須從左走到右，每個藥水你都可以考慮要不要喝，不過喝完藥水後生命值必須為正。問你最多可以喝幾罐藥水？

優先隊列 - Priority Queue

Codeforces 1526C2 - Potions (Hard Version)

一開始你的生命值為 0，由左至右有 n 個藥水。你必須從左走到右，每個藥水你都可以考慮要不要喝，不過喝完藥水後生命值必須為正。問你最多可以喝幾罐藥水？

這個問題基本上也是貪心問題，不過結論也還算簡單

優先隊列 - Priority Queue

Codeforces 1526C2 - Potions (Hard Version)

一開始你的生命值為 0，由左至右有 n 個藥水。你必須從左走到右，每個藥水你都可以考慮要不要喝，不過喝完藥水後生命值必須大於等於 0。問你最多可以喝幾罐藥水？

這題的想法很簡單，如果我們要喝第 i 個藥水時，我們必須要先維護我們的生命值要是非負的，那代表我們在喝這個藥水前，所有藥水的總和必須是非負的

優先隊列 - Priority Queue

Codeforces 1526C2 - Potions (Hard Version)

一開始你的生命值為 0，由左至右有 n 個藥水。你必須從左走到右，每個藥水你都可以考慮要不要喝，不過喝完藥水後生命值必須大於等於 0。問你最多可以喝幾罐藥水？

因此我們可以維護一個 Min Heap，當我們的總和不到 0 的時候，就把最小的元素拿掉。

優先隊列 - Priority Queue

Codeforces 1526C2 - Potions (Hard Version)

一開始你的生命值為 0，由左至右有 n 個藥水。你必須從左走到右，每個藥水你都可以考慮要不要喝，不過喝完藥水後生命值必須大於等於 0。問你最多可以喝幾罐藥水？

因此我們可以維護一個 Min Heap，當我們的總和不到 0 的時候，就把最小的元素拿掉。

時間複雜度: $O(n \log n)$

優先隊列 - Priority Queue

可以把 Heap 想像成是一種自動排序機，能夠在 $O(\log n)$ 的時間給你極值

因此有一種排序方式被稱為 Heap Sort，將所有數字都丟進 Heap 依序把最大或最小值拿出來，就完成排序了！

時間複雜度: $O(n \log n)$

集合 - Set

正如數學中的 Set，他的意義就是有不同的數字放在這個集合中
在 C++ 當中的 Set 是由一種我們稱為「紅黑樹 (Red-Black Tree)」實作的

他裡面的元素不能重複，而且元素從小到大排序好了

在 C++ 中方便的地方是可以直接做二分搜

集合 - Set

```
set<int> s;
```

- 宣告方式: set< 資料型態 > 變數名稱
- 加入元素: insert(x) (x 為要加入的元素)
- 把特定值的元素: erase(x) (x 為要移除的元素)
- 找最小的元素: *s.begin()
- 找最大的元素: *s.rbegin()
- 查詢大小: size()
- 是否為空: empty()
- 清除元素: clear()
- 二分搜: lower_bound(x)

Set 也是我們非常常拿來使用的一種資料結構，因為他能簡單幫助我們做二分搜

以下用 APCS 的例題來看看 Set 的方便之處吧!

APCS 202101 p3 切割費用

給你一根長度為 L 的棍子，以及 n 次刀切的位置，每次把棍子切半，所需的費用為棍子的長度，問總共要花多少費用

APCS 202101 p3 切割費用

給你一根長度為 L 的棍子，以及 n 次刀切的位置，每次把棍子切半，所需的費用為棍子的長度，問總共要花多少費用

這個問題，其實沒那麼困難，但我們要知道要如何維護刀切的棍子的長度，我們可以用 set 紀錄切過的位置，而棍子的長度即為目前要切的位置的前後相減

集合 - Set

```
int n,l;
cin >> n >> l;

set<int> s;
s.insert(0);
s.insert(1);

vector<pair<int,int>> v;
for(int i = 0;i < n;i++){
    int x,y;
    cin >> x >> y;
    v.push_back({y,x});
}

int ans = 0;
sort(v.begin(),v.end());

for(auto [y,x] : v){
    auto itr = s.insert(x).first;
    ans += *next(itr) - *prev(itr);
}

cout << ans << "\n";
```

雜湊表 - Map

下一個要講的東西是 Map，不過我不知道他的中文叫做什麼
不過他事實上是一種 Hash Table，就是你可以把字串、數字等等放進去
對應到另外一個不同的東西

雜湊表 - Map

```
map<int,int> m;
```

- 宣告方式: map< 資料型態, 資料型態 (二)> 變數名稱
- 查詢是否有某個 key: count(x)
- 找 key 對應到的值: m[x]

雜湊表 - Map

C++11 後可以這樣去枚舉 map 中的所有元素

```
for(auto p : m){  
    int key = p.first, cnt = p.second;  
}
```

雜湊表 - Map

C++17 後可以這樣去枚舉 map 中的所有元素

```
for(auto [key,cnt] : m){  
  
}
```

雜湊表 - Map

經典問題

輸入一個 n 項的陣列，接著有 q 個詢問，每次問 x 在陣列中出現幾次

雜湊表 - Map

經典問題

輸入一個 n 項的陣列，接著有 q 個詢問，每次問 x 在陣列中出現幾次

這題就可以輕鬆用 map 解決

雜湊表 - Map

有很多人可能不知道我出在時間複雜度的第二題要怎麼做
而他可以用 map 來做
程式碼