競賽導論及重要技巧

baluteshih

2019 年 9 月 11 日

1 前言

歡迎加入板中資訊的家庭，板中歷屆在資訊競賽上的成績不能說是很好，但也不能說是很差，歷屆出過國手這回事還是真實存在的。
然而在經過長久的努力後最近總算又有點成績出現了 (?)，還請各位對資訊競賽有興趣的學弟妹們，多多支持一下板中資訊，讓板中的資訊競賽能夠繼續傳承下去！

資訊不像是一般學科，儘管都是做題目，但仍然有著與其他學科的差別在。
其最明顯的差異就是 ── 即時回饋！
扣除掉一些少數的後測比賽，大多數的比賽都可以讓你在寫完程式後，立刻送出程式，立刻回傳裁判結果，有些比賽還有外部的記分板供外部觀眾觀賞刺激的比賽直播，選手們在比賽結束後通常也能立即得知自己的狀況。
最公平的是，資訊競賽不存在「評審的看法」，測試資料都是事先就完成的，除非測資本身出錯，不然在大家都用同樣測資的情況下，絕對不會出現因為哪個評審覺得如何就特別高分的情況！
還在等什麼？快繼續往下看吧！

2 你應該要知道的事

2.1 Online Judge

講了那麼多，資訊競賽到底要如何練習呢？
最直接的方法就是上網找線上的評審系統 (Online Judge)，由於資訊競賽漸漸的抬頭，網路上已經囤積了無數道題目可供各位練習了，各位如果都在這個時代進入競賽世界了，何不好好利用一下資源呢？

Table 1: 常見 OJ 一覽

簡稱	網址	簡介
ZJ	zerojudge.tw	板中最常用的 Judge，據說號稱「水題庫」
TIOJ	tioj.infor.org	建中的 Judge，在這裡刷題可以磨練 coding 的能力和耐心。 你說你不懂什麼叫磨練耐心？去寫寫就知道了。
UVA	uva.onlinejudge.org	號稱最老的題目庫，有大量的相同性質題目可以刷，缺點是常常會有題目的輸入輸出難度大於題目本身難度，非常難用 (X UVA 其實是老名字了，現在他的名字叫做「Online Judge」，可是很蠢所以只能拿來玩梗。
CF	codeforces.com	一個俄羅斯的 Judge，可以看到他人的 code，不錯的觀摩點。 不定期出現線上比賽。 有個精湛的點是他可以爬積分，而且 Judge 很優質，大家都很喜歡他。
Atcoder	beta.atcoder.jp	日本的 judge，每個禮拜六定期都有比賽，一樣有積分爬，而且時間很適合台灣人。
51Nod	51nod.com	一個大陸 Judge，因為題目都是中文的很好閱讀，也常常從其他 Judge 抓題目過來翻譯，缺點是 Judge 執行速度慢。
vjudge	vjudge.net	擁有從各大外國 judge 抓題目進來的能力 (不過同時表示題目幾乎都是英文的)。 其最大的優點是任何人都可以用 vjudge 開比賽，甚至可以拿各大 judge 的題目混在一起出比賽。

網路上其實還有很多 judge 可以利用，有興趣可以自己上網找找。

而其中板中的教學比較常會用到「TIOJ」和「ZJ」，所以還請各位早早先去註冊這兩個網站，順便找些題目來試用看看吧！

2.2 解題

在解題的過程中，有些知識是不能不知道的。

Table 2: 常見 judge 結果

結果	意義
AC (Accept)	通過了！歡呼 OwO！
WA (Wrong Answer)	答案錯誤，你一定寫錯了。
TLE (Time Limit Exceed)	超時，有可能是演算法上使用錯誤，也有可能是不小心打到 while(1) 之類的 (?
MLE (Memory Limit Exceed)	記憶體超出限制，請使用空間複雜度較低的演算法 (遞迴過深也會出現類似情形)。
RE (Runtime Error)	讀到不該讀的記憶體，據說 1/0 這種除法錯誤也歸類在這裡。
CE (Compile Error)	你不會用 Compiler？

以上一般大比賽常使用的 judge 結果，當然也有些 judge 會出現不同的結果，還是以競賽時的說明為準較好。

有些比賽的題目其中會有部分分數，有些則沒有 (?
遇到部分分數的題目時，如果沒有萬全的把握可以把整題拿下來，建議先穩穩把一些部分分數抓下來，至少確保結算之後的成績不會很難看。

通常比賽的排名方式會是分數優先，再來是比較時間 (penalty)。
時間上的計算通常會以第一次 AC 每一個題目的經過時間做加成，有些比賽會在程式執行出現非 AC(CE 除外) 的結果時給予 20 分鐘的 penalty，所以並不是一直愚笨地狂送 code 就可以了。

再來講一下輸入輸出的部分，在某些題目中，會遇到測資量極為龐大的問題，有時候題目會專門卡住你的輸入輸出，害你狂吃 TLE。
這時候有一個最直接的解決方法，就是在 main 函式裡的第一行直接加上：

```
1 std::ios::sync_with_stdio(false), std::cin.tie(false);
```

這裡不方便解釋他的實際用途，但有個使用條件，當你加了這一行後，所有關於 C 的輸入輸出都必須被捨棄，這牽扯到 C++ 的 cin、cout 等相容性的問題，但是可以保證加了它之後，cin、cout 絕對能快上好幾倍 (在大量測資上可以看出差別)。

如果真的很好奇也可以去看看以下網站：
http://chino.taipei/note-2016-0311C-的輸出入 cin-cout 和 scanf-printf 誰比較快？/
還有就是，加了那行之後，所有的輸出會在程式結束後才會一口氣貼上來，所以如果你想動態看輸出記得要註解掉，不然會很崩潰 ww

2.3 習題

你問我為什麼這裡有習題？看了題目就知道了！

1. (TIOJ 1818) 請你仔細閱讀題目的應考須知，並完成題目。

3 複雜度？

在資訊競賽上，分析演算法和資料結構 (DSA) 的複雜度是一件很重要的事情。
這裡先介紹最基本的常用函數。

3.1 Big O notation

又稱大 O 函數，可以用來表示一個複雜度的「上界」。
至於定義是什麼呢？先上維基百科的定義。

$$\text{定義 } f(n) \in O(g(n)) \text{ 若且唯若 } \exists c, N \in \mathbb{R}^+, \forall n \geqq N \text{ 有 } |f(n)| \leqq |cg(n)|$$

簡單的說法是，若 $g(n)$ 的成長速率優於 $f(n)$ 的話，我們就說 $f(n) \in O(g(n))$。
不懂嗎？相信看了以下的例子你就懂了：

$f(n) = 87n^2 + 31415926n + 7122$，則當 n 很大的時候，可以想像 $31415926n + 7122$ 會遠小於 $87n^2$，而 87 又是一個常數，我們可以說 $f(n) \in O(n^2)$

也就是說，在分析一個函數的複雜度時，找到影響規模最大的項目，就可以大概抓到這個函數的複雜度上界。
這也是最常被拿來在資訊競賽上計算複雜度的工具，往後會不斷地提到它。

3.2 Big Ω notation

有了上界就有了下界，這裡就不嚴謹定義它是什麼東西，只是單純希望你們能夠在看到它的時候可以知道它是什麼意思。
最簡單的講法是，對於任意的 $f(n) \in O(g(n))$，都有 $g(n) \in \Omega(f(n))$。
大 Ω 函數是估算複雜度下界的工具，若有歸約法的章節它可能會再出現 (?)。

3.3 Big Θ notation

這東西是最嚴格的函數，也就是要同時滿足 $f(n) \in O(g(n))$ 且 $g(n) \in \Omega(f(n))$，我們才可以說 $f(n) \in \Theta(g(n))$。
一樣是暫時不常見的函數，就不再多說。

3.4 時間複雜度

終於來到正題了，在資訊競賽上我們最重視的幾乎就是 DSA 的時間複雜度，你可能會一時轉換不過來，因為我們剛剛都還糾結在數學上的定義。
對於每一份寫出來的 code，我們把所有的運算都當成花費一單位時間 ($O(1)$)。(這裡的運算指的是 +-*/%、甚至是 ==、> 這種比較運算子，我們都稱為一個運算。)
最後把所有的迴圈、遞迴函數、輸入輸出 (也當作 $O(1)$)⋯等等的操作統合起來，就可以得到一個屬於這份 code 的時間複雜度函數 $f(n)$，進而由輸入 n 的範圍去估計這份 code 有沒有可能獲得 TLE。
不過需要注意的是，除法 (/)、模運算 (%) 的常數其實比一般運算都還來得大，尤其是後者，據說有著 7 倍的差距。
像這種常數大的工具不只在這兩個運算上會看到，往後其他的算法上也有可能遇見常數大的問題，初學競賽也許不會，但在後面的題目常常會出現明明複雜度 OK 但因為常數大就吃了 TLE 的問題，這時候我們就會用到一個叫「壓常數」的技巧，不過還是先別在這裡提了。

如果不是很了解的話，這裡舉個例子：

Algorithm 2: Bubble sort

```
1 void Bubble_sort(int *arr,int n){
2     for(int i=0;i<n;++i)
3         for(int j=1;j<n-i;--j)
4             if(arr[j]<arr[j-1])
5                 swap(arr[j],arr[j-1]);
6 }
```

這是氣泡排序法的演算法，從裡面我們可以看到使用到 if 的次數約是 $n(n-1)/2$，所以演算法的時間複雜度就是 $O(n^2)$。

3.5 空間複雜度

一份 code 所宣告的記憶體量便是所謂的空間複雜度，有時候記憶體宣告太多會導致你的程式碼獲得一個 MLE，這時候就表示你的空間複雜度爛掉了。
空間複雜度比起時間複雜度來得好估計，只要了解 byte 到 MB 之間的轉換，搭配題目給的記憶體限制的話，相信這不是一件難事。

3.6 常見時間複雜度

估出一個時間複雜度後，你可能會有疑惑，到底這樣的複雜度會不會 TLE。
這裡我就給出一些常見複雜度上的比較，由於我們比賽常用的語言是 C/C++，你可以先假設電腦一秒能夠執行 10^8 次運算，並且適當的考慮自己演算法的常數，剩下的就都是經驗問題了。

$$O(1) < O(\log\ n) < O(n) < O(n \log\ n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

可以自己試著代數字進去看看，順便體會一下各種複雜度的成長速率。
當然並不是所有的複雜度都可以歸約到以上情況，至於還有什麼類型的複雜度將會在以後的算法上遇到，這裡就不一一列舉。

3.7 均攤複雜度

均攤複雜度是一個很神奇的概念，最常見應該是在圖論的演算法上，甚至在其他 DSA 的優化都會出現。
在一個算法裡面，雖然可能有多層迴圈在運作，但如果我們可以保證這些迴圈跑到的次數最後總合起來會是一個比較正常的複雜度的話，就可以說明你把複雜度「均攤」掉了。
屆時遇到我們會再說明清楚這個概念。

4 到底有哪些競賽能比？

如果只是想練比賽的話倒是不用擔心，第一節我就有提到其實網路上的 OJ 常會有比賽可以練習。
資訊競賽的種類其實非常繁多，不過這裡還是列舉一下常見的大比賽們，還有高中資訊競賽的主線。

4.1 主線比賽

Table 3: 主線比賽

時間	比賽名稱	簡介
9 月	校內資訊學科能力競賽	校內初選，由於在開學之際所以常常會使一年級的學弟妹 QQ 掉。 但是近三年似乎都沒這個情況 (?
11 月	新北市資訊學科能力競賽	新北市的比賽，基本上就是板中的主場 (X 不過可別輕視這場比賽，它是進入全國賽前的門檻，而且又是很討厭的賽後 judge，一不小心失誤的可能會導致眾多悔恨 (? 得獎也別太高興，可以到全國再說話。 前七名可以晉級全國賽。
12 月	全國資訊學科能力競賽	全國的各區賽上來的好手都會聚集在這裡，前十名可以直接進入 TOI 一階。 值得一提的是，常常會有大神在市賽雷掉或是因為有更多大神而上不來全國賽，所以通常全國賽的排名會比一般大比賽來得優秀。
3 月	TOI 初選	會再選出 20 人進入 TOI 一階。 主要由師大舉辦，曾經有 Judge 爆掉導致整場比賽變成無限 submit 後測大賽的經驗。
4 月	TOI 一階	培訓 14 天，會有兩次模考，最後選出 12 人進入 TOI 二階，並參加 APIO。
4 月	TOI 二階	再培訓 14 天，一樣兩次模考，最後選出 4 人成為正式國手參加 IOI。
5 月	APIO	亞太資訊奧林匹亞競賽，得獎的話似乎只能獲得榮譽感 (?
7 月	IOI	高中資訊競賽的最終目標，國家這年的希望就在你身上了。

4.2 次要比賽 (怎麼都是團體賽)

Table 4: 主線比賽

時間	比賽名稱	簡介
8 月	ISSC	電腦學會舉辦的青年程式競賽，英文出題，三人一隊報名。
12 月	NPSC	同校至多三人組隊報名，每校至多三隊進入決賽，獎品豐富，有分國中組與高中組。教育部認可的全國性競賽，獎狀有一定可信度。
11 月	CodeWars	一樣是三人一隊的團體賽，利用大量的題目來輾壓選手，防止選手破台，但是比賽結束會有大抽獎，大獎是筆電，板中的紀錄是只要有參加就有人抽中。 不過還是有怪物能破台。
7 月	成大高中生程式邀請賽	三人一隊的團體賽，題目難度普通，有點適合拿來練團隊默契。
4 月	Ytp 圖靈計畫	2016 才出現的新興比賽，有分三個階段，只有第一階段是純三人一隊的比賽，後兩個階段將會進行資訊相關的技術培訓 (重點是有提供獎學金)，如果有空閒時間可以考慮看看。

其實資訊比賽不僅僅只有這些，當然如果有能夠讓學生出去爭取榮譽的機會，板中也會盡全力讓各位出去比賽的。

4.3 程式設計檢測？

據說有個東西叫做大學程式設計先修檢測 (APCS)，不過裡面的題目都不會很難，難的地方在於它是完全的賽後 Judge，考驗的除了是細心度之外，也是對於基礎題目的熟練度。相信對於打算把資訊競賽摸熟的各位，APCS 絕對不是什麼會難倒各位的檢測。

5 競賽重要技巧及概念

有些重要概念是很難當成一個章節來解釋的，以下提到的這些通常一時間都無法完全理解，要等以後開始實作才會慢慢體悟的。
由於這些概念不懂的話很容易在往後的課程炸爛，所以這裡先特別提出來做個宣導 (?)

5.1 指標（Pointer）

接觸 C 語言時多少都有聽過不少人在抱怨「指標」有多困難、多難搞。
實際上如果不小心學錯了方向，指標的確會成為學習者的噩夢。
但是在競程上，想寫好資料結構指標卻是不可或缺的工具之一，為了學習順利不得不先打好指標的基底才行。
因為這東西太容易爛掉了，我怕我講得不是很好，在這裡推薦一篇文章供各位參考 (X

https://codingsimplifylife.blogspot.com/2016/06/c-12-pointer.html

如果你想問使用時機的話，通常在競賽上會出現的就是「動態記憶體配置」

```
1 int *p=new int;
```

以上是 C++ 的例子，可以新增一個記憶體當成 int 並由對應的指標 p 負責記錄其位置，主要在往後學到樹型資料結構時就會遇到了。
你說你不知道樹型資料結構是什麼？簡單說就是變數之間連結的關係會長得很畸形，導致你沒辦法一口氣把記憶體宣告好，現階段你只要了解到之後會這樣用就行了。
最後別忘記釋放他。

```
1 delete p;
```

5.2 參考（Reference）

在宣告一個變數時，在變數名前加上一個「&」字元，你就可以得到這個變數的參考型態。參考型態代表了變數或物件的一個別名，可以直接取得變數或物件的位址，並間接透過參考型態別名來操作物件，其作用類似於指標，但卻不必使用指標語法，也就是不必使用 * 運算子來提取值。
使用方法如下：

```
1 int &x=arr[i][j][k];
```

由於是取得變數位址，所以參考型態的宣告絕對要初始化到其他變數上。
特意在後面放一個三維陣列的用意是讓各位理解到他的用處，在撰寫程式碼的時候難免會

需要重複使用好幾次同一個變數，由於該變數可能是一個陣列的元素，導致其變數名過於冗長，但是也不能宣告一個新的變數來記錄其值，因為可能還得改值！
這時候參考就派上用場了，宣告參考等同於在幫該變數取綽號，在其有效宣告範圍內，C++ 都會幫你視為同一個變數的。
還有一個很大的用處是這樣：

Algorithm 6: Swap function

```
1 void swap(int &a,int &b){
2     int t;
3     t=a,a=b,b=t;
4 }
```

傳參考進函數的話，就不需要動用指標了！是不是很方便呢？
當然他也不是能取代掉指標的等級，兩者是有一定差別存在的。
往後寫資料結構的時候絕對能體會到他的方便，請拭目以待。

5.3 遞迴 (Recursive)

大多數程式語言的函式都具備著遞迴呼叫的功能，也就是在函式內重複呼叫自己本身。
撰寫遞迴函式時永遠都要考慮以下問題：

是否存在一個終止遞迴的條件。

一個高中數學常提到的遞迴函式例子：

Algorithm 7: Fibonacci number

```
1 void F(int x){
2     if(x==0 || x==1) return 1;
3     return F(x-1) + F(x-2);
4 }
```

想想看如果把第二行拿掉會發生多恐怖的事情？
函式會無限往下呼叫直到程式炸爛！
由於遞迴函式在程式內部是用堆疊實作的，一個函式也同時代表著一些記憶體，無限呼叫就會造成各位耳熟能詳的「stackoverflow」。
現在的程式都足夠聰明會去阻止這件事發生，並回傳你一個程式停止回應，但其本身是非常危險的事情，還請各位多加注意。

撰寫一個遞迴函式最重要的思路大概如下：

1. 以「遞迴後得到的東西是對的」的前提下寫程式。
2. 設置好基底。

以最經典的遞迴問題－－河內塔為例

Description

有 A,B,C 三個圓柱，請你輸出 N 層河內塔問題的解，使得 N 個圓盤能從 A 圓柱搬到 C 圓柱上。

讓我們假設 $solve(int\ n, char\ A, char\ B, char\ C)$ 是解決河內塔問題的遞迴函式。
$Move(int\ n, char\ S, char\ T)$ 是把 n 號圓盤從 S 移動到 T 上。

Algorithm 8: Tower of Hanoi

```
1 void solve(int n,char A,char B,char C){
2     if(n==1){Move(n,A,C);return;}//基底
3     solve(n-1,A,C,B);//先假設我們已經把n-1個圓盤從A搬到B了
4     Move(n,A,C);
5     solve(n-1,B,A,C);//再假設我們把已經搬到B的n-1個圓盤搬到C身上
6 }
```

如此以來思路是不是變得很清楚了呢？
遞迴的相關用途也一樣在往後會不斷的重複提到，還請各位務必要熟悉此概念。

5.4 建構子 (Constructor)

通常在自定義 struct 的時候，往往會想要宣告一個變數並再塞一些值進去。直覺想法就是宣告一個空的變數之後再把值放進去，但這樣顯然很浪費時間。以下提供一點方法：

Algorithm 9: Constructor

```
1 struct myst{
2     int x;
3     double y;
4     myst():x(0),y(0){}
5     myst(int _x,double _y):x(_x),y(_y){}
6 };
```

冒號後方的是成員預設清單，可以在裡面放任何對於該型態內有的變數要賦予的值。你也可以在大括號裡面執行一些程式，好比迴圈建表之類的，簡單來說一切都是為了讓你方便初始化用。
上面的程式碼寫了兩個建構子，一個有傳變數進去，一個則沒有，呼叫兩者的方式分別就是 myst() 和 myst(x,y)，這兩個在宣告完建構子之後就可以直接被當成一個變數值用了。前者會被 C++ 認定是「預設建構函式」，也就是宣告在全域時會幫你把所有變數全部依照這個函式預設。但需要注意的是，其實原本編譯器就會幫你寫好預設建構函式，預設就是對內部所有變數呼叫他們的預設建構函式，可是一旦你宣告了任何非預設建構函式，編譯器就不會提供預設建構函式了，所以要額外自己寫。

5.5 自定義運算子

在撰寫一些算法時，可能對於某些型態 (尤其是自定義的 struct) 必須反覆的使用同一種運算，這時可以賦予這個型態自定義的運算子，來大大減少 coding 的時間，也可以獲得一點程式上的加速。

Algorithm 10: operator 1

```
struct myst{
    int x;
    myst(int _x):x(_x){}
    bool operator<(const myst &a)const{//小於
        return x<a.x;
        //在<左邊的是「呼叫運算子的變數」，所以不用變數名稱
    }
    myst operator+(const myst &a)const{//加法
        return myst(x+a.x);
    }
};
```

格式就如同上面所寫，請根據你要的運算子給予相對的回傳型態，然後確保不會沒有回傳值，不然會爆炸。

如果怎麼樣都無法接受這種寫法的話，說不定以下的寫法會讓你比較好理解：

Algorithm 11: operator 2

```
struct myst{
    int x;
    myst(int _x):x(_x){}
};
bool operator<(const myst &a,const myst &b){
    return a.x<b.x;
}
myst operator+(const myst &a,const myst &b){
    return myst(a.x+b.x);
}
```

寫在外面是允許的，不過你要清楚的把兩個變數名稱寫進去。

5.6 習題

1. (ZJ a227) 河內塔
2. (ZJ a024) 最大公因數，請試著用遞迴寫出輾轉相除法。
3. (TIOJ 1994) 題敘略。