April Fools Day Contest 2024

adrien1018, FHVirus

首先感謝大家的參與（這次是目前最多人參加的一次 April Fools Contest！），不知道大家被整得還開心嗎 (#

pA. 不祥之題

April Fools Contest 終於開賽了，趕快來傳第一題！然後……恭喜你得到了 -20 分 (?

沒錯，這題就是很純粹的整人題，而且不只是拿到 -20 分的部分，還有其實根本就沒辦法拿到除了 -20 分以外分數的部分！畢竟題目也已經講得很清楚「做過這題的人都會遭遇一定程度的厄運」，那唯一能**避免**這個詛咒的方法自然就是不要做這題囉！傳 CE、CLE submission 是個不錯的嘗試，special judge 也會在 CE 給你顯示 87 分、CLE 會顯示 100 分，但是 CE/CLE submission 本來就是不計分的 :)

題外話，這題是這場比賽「逆封板」的部分原因（另外一部分是避免 scoreboard 把風向帶得太偏），不過操作失誤的關係開賽前一兩分鐘忘了封板，希望沒有造成太大的影響。這題也是這場比賽設定成註冊制的原因，因為只要你有註冊這場比賽，就算你一個 submission 也沒傳，你也不會是最後一名喔！

pB. 🟦🟦🟦💥1️⃣2️⃣🟦

這題是本場比賽最難發現梗的一題。整題給的提示非常少，除了標題看起來很奇怪、測資範圍也有點玄之外，好像沒有什麼太有用的資訊，就算丟提問系統也只會得到由方格組成的回覆。既然如此，這些方格到底有沒有什麼意義呢？

這題的難點就是要從方格聯想到由方格組成的遊戲，具體來說是 Minecraft。如果有想到 Minecraft 的話應該就相對簡單了：標題中的火花圖案可以聯想到 Minecraft 的爆炸，然後方塊裡面有計數感覺跟爆炸可以傳多遠有關，再從範測中的數字去 Google（或者直接看 Minecraft Wiki 的 Explosion 頁面）不難發現 output 的那些數字是不同方塊的 blast resistance。不過即使沒有聯想到 Minecraft，如果你有想到可以直接把 Sample Output 整個拿去 Google，第一個搜尋結果也是一個跟 Minecraft blast resistance 相關的 code！

接下來就是搞清楚每個數字代表的是什麼方塊。題目中的數字「12」同時也代表著

Minecraft 的 1.12 版本，而這也是數字 block ID 存在的最後一個版本（後來就改成字串了），而 1.12 的 block ID 最大也是到 252，因此結論就是對於 1.12 版本的每個 block ID 輸出它的 blast resistance。

但是還有一件有些麻煩的事：想辦法找到每個 block 的 blast resistance 到底是多少。Minecraft Wiki 裡面並沒有完整的資訊，而且要手打這麼多數字也很累。不過剛好網路上可以找得到 1.12 版本的 source code，然後再 Google 一下可以發現 blast resistance 的設定在 `registerBlocks` 函數裡面，所以就簡單的 parse 一下就可以了。需要注意的是這裡 `setResistance` 的值要乘以 0.6 才會是正確的 blast resistance，而 `setHardness` 的值是正確的 blast resistance，同時前者會覆蓋掉後者。第一筆小測資也可以用來檢驗有沒有找到正確的值。

這題當初出題時其實是因為出完不少題目之後發現好像都沒有範測找規律題，所以決定還是塞一題，但因為 CF 的 April Fools Contest 的範測找規律題通常都偏簡單，就反其道而行出了一題很難的 (?

pC. 隱藏

這題是一題蠻直白的題目，輸出要求的三行就是對應三筆測資，每答對一筆測資就會同時得到下一筆的答案。第一筆最簡單，就是把答案單純用白色的字隱藏起來。第二筆會發現有訊息但裡面看起來是空的，不過如果用 F12 inspect element 或者查看原始碼的功能看一下就會發現那其實是一個空的 div，而 div 裡面有個答案的 attribute。到了最後一筆會發現 TIOJ 居然壞了 (?) 不過仔細想一下可以想到這是用 JavaScript 把整個網頁換成 TIOJ 的 500 (Internal Server Error) page，所以答案應該就在 JavaScript 裡面，用 F12 看 request、disable JavaScript、重新整理之後查看原始碼（如果是在還沒 judge 完之前進入 submission 頁面，judge 結果會是用 websocket 送的，所以檢視原始碼看不到，要用 F12 看 request 才行）等方式就可以拿到答案。

這題有個小彩蛋[1]是那個 500 page 是出題者直接 invoke 500 然後把整個網頁的 HTML 拿來用，所以右上角的 username 是出題者的 username，另外 About 的連結也是錯的。不過後來發現的時候想說這也是個不錯的提示，就把它留著了。另外有不少人在看到 500 page 的時候就來提問，不過大部分的人在我們回覆之前就自己先發現梗了 xd

pD. 以牙還牙

傳了一個 submission 就被 judge rickroll 了:) 那既然標題是以牙還牙，應該就是 rickroll 回去就對了，但具體來說怎麼 rickroll 回去呢？

[1]其實是出題的時候沒有注意到 (?

這題的關鍵是「以牙還牙」同時也代表著 rickroll 回去的時候要某種程度上是「對等」的，所以既然 judge 用來 rickroll 的影片是 *Never Gonna Give You Up* 的第一節（含副歌），那應該如數奉還才符合對等關係，所以就是把這個部分的歌詞 print 出來就能 AC 了。

一個小細節是 print 歌詞的時候還是要依照正確的輸出格式（輸出「一行」字串）才能拿到 AC。不過如果沒有這麼做的話會發現 judge 就不會丟 rickroll 給你了（因為 judge 是先判斷輸出格式是否正確），也算是一個小提示。另外這題的 special judge 是判斷把標點符號拿掉、字母轉小寫之後 LCS（substring）超過 100 且 LCS（subsequence）超過 280，所以有沒有標點符號或者歌詞少部分的不一致（例如有些歌詞把 (do I) 的回音打出來）不會影響 AC。

這題的提問都單純都是用 *Never Gonna Give You Up* 的影片加上字幕回覆，沒有什麼提示功能。

pE. 經典題

這題就是非常標準的 error correction code 題目，如果對 error correction code 有一些了解就知道這可以用標準的 $\mathrm{GF}(2^8)$ 下的 Reed-Solomon code 實現。網路上有不少 Reed-Solomon code 的實作，照著做就可以拿到 AC 了。

當然這麼複雜的演算法肯定不會是愚人節比賽的正解，因此需要好好的研究一下這題的分數的計算方式。可以發現由於是用每筆測資平均的方式計算的，所以可以發現就算用很爛的方法傳送，只要把數字全部集中在一個序列當中全部傳送出去、剩下的序列隨便 output 也可以拿到接近滿分的分數！考慮最簡單的 error correction 方法：傳送的時候把每個數字重複很多次變成一組數字，接收的時候從每一組數字裡面挑眾數當作答案。這樣如果替換是隨機發生的，那麼可以預期只要重複夠多次，同一組裡面有足夠多數字被替換造成接收到錯誤的數字的機率就會足夠小：如果重複 $M/N = 7$ 次的話，一個數字的錯誤率大約是 $p = 7 \times (1/10)^6$（這一項會蓋掉其他項，因為被替換成同樣數字的機率更小），考慮最多總共有 1500×129 個數字（序列長度也要傳），可得全部正確的機率有 $1 - (1-p)^{1500\times129} \approx 25.8\%$，而且實際上測資沒有每個都是 128 個數字，所以試幾次（例如改變一下 output 的順序）就可以 AC 了。

不過這題題目並沒有保證替換是隨機發生的。事實上，這題的 special judge 會在某些測資刻意挑連續的數字替換、或者在替換時全部換成同一個數字。但這個也很好解決：前者可以把所有數字 random shuffle 之後傳送、接收的時候再用逆排列轉換回來；後者可以在傳送前把每個數字都在 mod 256 底下加上某個 random 的值，收到的時候減回來即可。

不過這個東西實作起來也不簡單，而且還要分析機率，還是不怎麼像愚人節比賽的題目？其實這題還有一個更好寫（但比較難找到梗）的做法，關鍵在於題目中兩個提

示：一是題目強調「出題者很懶」，二是看起來不怎麼尋常的範例測資。一個很懶的出題者可能會怎麼生測資呢？可能是直接用 random 生測資然後也不設 seed 吧 (?) 所以其實只要寫一些最直覺的 random 生測資程式，然後跟範例測資對照一下，就可以發現用 `std::ranlux48` 生出來的測資會跟範測一模一樣。事實上，assert 一下測資就可以發現 99 分 subtask 的第一筆測資跟範例測資完全一樣，可以做為驗證想法的工具。也就是說只要每筆測資隨便傳個東西（還是要傳一個數字因為題目規定 M 是正整數），然後第二次執行的時候跑生測資的程式就可以 AC 了！

其實最開始出題的時候只有打算給這個做法，不過後來在寫完分數計算方式的時候發現了用平均計算會有「漏洞」，也覺得這個拿來出題蠻有趣的，所以就多出了這個分數計算的梗。這題的提問系統也沒什麼提示，不過回覆的某些字元會用題目所說的方式替換掉。

pF. 錯誤

這題很簡單明瞭，要的就是「錯誤」。如果你輸出了「正確」的答案你會獲得 AC 0 分，但如果輸出錯誤也只會拿到 1 分，然後問你「有沒有辦法再更錯誤一點」?

有什麼是比錯誤的答案還要錯的東西呢？大概就是連 code 就是錯的吧！於是只要隨便上傳個 CE 的 code 就可以拿到 AC 了。接著會發現分數是個奇怪的數字，如果多傳幾次可能會發現好像 CE 訊息愈長、code 愈短就會拿到愈高的分數。事實上，這題的分數計算方式就是 $\frac{650\ln(x+1)}{100+y}$，其中 x, y 分別是 CE message 長度和 code 長度。也因此，這題的目標就是寫出 CE message 最長的程式。Google 一下不難找到這個 stack exchange 上的挑戰，裡面就有不少可以嘗試的東西[2]。不過因為 TIOJ 編譯有 4GB 的 memory / output limit 和 60 秒的時限，所以還要讓 error message 產生得夠快才行。實驗一下可以發現最好的方法是兩行 `#include __FILE__`，這樣大概可以拿到 104 分左右。另外因為 TIOJ 在編譯時檔名是 `prog.*`，所以如果改用 C 傳的話 `#include"prog.c"` 可以讓 code 短一些多拿一點點分數。這題也自然成為了對 judge DoS 的題目，還好 submission 數量沒有太多所以還撐得住，本來還想說如果大家太喜歡傳這題的話要把 CD 開大 (?

這題還有個小彩蛋是既然是「錯誤」，所以 sample input 也刻意弄成錯的，不過不影響作答就是了。另外這個可以對 submission 總結果做奇怪的事是 TIOJ 最新版本（3.1.0）的功能，一部份的原因就是為了這場比賽。然後這次 TIOJ 的 judge 也一度被打壞了[3]，原因就是因為寫這個新功能時忘記考慮非 UTF-8 的字串。

[2] 如果要在自己的電腦上實驗的話要特別注意，裡面一些會讓 compiler 用超多記憶體的程式很有可能會直接讓電腦死當。
[3] 話說出題者以前也曾經用 compile 輸出一堆 error message 同時用一堆記憶體的 code 打壞過別人的 judge (?

pG. 直式乘法

你有看過直著寫的題目嗎？

做法其實很簡單，就是把兩個矩陣讀進來然後相乘輸出。寫完之後上傳會得到 TLE/RE，再看看直著寫的 Hint「你有辦法**這樣讀**字嗎？」猜一下發現輸入也要跟題目一樣直的一行一行讀，或是先把每一行讀進來後轉置完再做事。當然，輸出也要先轉成直的再輸出，畢竟出題者也把自己原地轉置了。不過寫完後上傳會發現 WA，這時候就需要通靈，把輸出的矩陣自己轉置一次再轉置整個輸出即可。

順帶一提，這題的提問也都會用轉置過的字來回答。而這題的直式乘法是故意寫錯的（第二個數字應為 442809），很高興有人看出來。希望他沒有走歪路。

最後，感謝 `Fysty` 和 `Wayne Yan` 提供出題想法和轉置原理教學。

pH. Déjà vu

題敘給了 $N, e, m,$ nonce, ct，一臉就是個 RSA 題？

Google 一下可以查到提示提到的 Cado-NFS 是個質因數分解的程式，加上又提到 AES，那應該就是用 RSA 來加密 AES 的 key。又因為題目有 12 byte 的 nonce、ct 分為兩個部分、第二個部分是 16 byte，看起來很像 authenticated encryption 的 tag，而這些都符合 AES-GCM 的特性。既然如此，這題的 N 沒有很大，Cado-NFS 大概 20∼60 分鐘就可以炸出質因數分解（端看用多少 CPU，自己實驗是 16 個 thread 可以 20 分鐘跑完），之後解 RSA 發現明文確實是 256 bit，所以就用例如 PyCryptodome 之類的 library 解密 AES-256-GCM（Google 可以找到有很多 online AES calculator，不過我試過的幾個都沒辦法好好設定 nonce，所以才說它們很爛。另外這裡要嘗試一下 key 的 byte order，用 little endian 才會是對的，可以用 tag 驗證正確性），最後解出明文會發現……咦？又是 rickroll？

你有看標題嗎？「Déjà vu」是「既視感」（好像在哪裡看過）的意思，那這題有哪個部分是「好像在哪裡看過」呢？Output Format「請輸出一行包含一個長度不超過 500 的字串。」是不是感覺好像在哪出現過？沒錯，就是 pD！其實不只 Output Format，TL/ML/OL 也和 pD 長得一樣。也因此這題 AC 的條件也跟 pD 一模一樣，輸出 *Never Gonna Give You Up* 第一節含副歌的歌詞就 AC 了。當然如果你 pD 沒有做出來的話這題就真的要 Cado-NFS 了，不過如果你成功解出這題的明文，有很大的可能性你會想到要把它丟回去 pD，這樣炸一次質因數分解 AC 兩題也還算是划算吧 (?

這題的提問系統則是會給你同樣格式加密的 $N, e, m,$ nonce, ct，可以從 ct 的長度判斷是哪一種回覆，而且 N 比較小所以 Cado-NFS 兩分鐘就能跑完，可以用來實驗加密

格式。另外三種回覆的 key 和 N 都會一樣，所以如果拿到兩種不同的回覆就可以用 common modulus attack 直接解出 key，不過因為跟原題的 key 不一樣所以拿到這些沒什麼用，反而還有機會走歪去解這個 key (?

pI. EZDSA

這題是為了掩護重要資訊被變成閱讀測驗的通靈題，希望有人看完 (?

首先，題敘的最後（實作細節上方）有說本題「以簽章的正確性為唯一的考量」，又題目定義「正確性」的意思是「被正確簽署的簽章總是要被驗證為正確的」（在 ECDSA 介紹第三行）。欸，那對亂簽的也輸出 `True` 的不也滿足這個限制嗎？

沒錯，只要全部都輸出 `True` 就好了！不過上傳之後會拿到一個不明就裡的分數和一則訊息「我要怎麼驗證這份程式碼的確是你寫的（雖然這是廢話）？」這句對照到題敘第一行「以上是本題的橢圓曲線數位簽章，可以用來證明本題的確是出題者出的（雖然這是廢話）。」可以發現上面的簽章莫名的用了 C++ 的註解格式，提示你要把這些加入程式碼的開頭才可以 AC。而 judge 也算在本題實作範圍，因此只管簽章的正確性，所以隨意貼一個格式一樣的簽章即可。

這題的提問系統也都會用和題目一樣的格式簽署，作為額外的提示，不過看起來有戳到提問的人都還是沒有發現這個梗。而那個不明就裡的 subtask 分數的公式是 $X \times \frac{5\ln(t)}{100}$，其中 X 是 subtask 總分、t 是執行時間。用 2 秒的時限代入可以得到 71 分左右，但由於這題的分數計算是用 wall clock time，所以如果用 `usleep(2950000)` 之類的跑到接近 3 秒的 wall clock limit 可以提高到 73 分（wall clock limit 可參見 verdict 說明）。

另外一個可能的做法是發現這題每一題都是單一的是非題，所以可以 assert 出每一個 subtask 到底是 True 還是 False，而很有可能做這件事的同時就不小心 AC 了 (?) 出題者其實有想到這個可能性，也有考慮要不要改成多筆測資，但後來想說就算是多筆測資可能也會有人不管三七二十一就先 `puts("True")` 再說，所以就乾脆也開放這個做法了。

感謝 `Fysty` 提供本題（基本上整題的）出題想法。

pJ. 1

標題、題目、Input、Hint 都寫 1，那應該就是輸出 1 囉，但是題目卻叫你輸出一大堆非負整數？

這題的梗其實就是在輸出格式。如果輸出 1 不對，有沒有可能「換個方式」輸出 1 呢？如果盯著 Output Format 看久一點，除了發現它是二維的格式外，或許還會發現每行

的數字量是行數的三倍。這其實是個提示：正確的輸出格式是一張圖片的 RGB 值[4]！至於是什麼圖片呢？當然是一張包含「1」的圖片囉！於是只要想辦法生出一張包含 1 的 24×24 的圖片然後把 RGB 值輸出出來就好了。這題的 special judge 很鬆，只要偵測到圖片中有一個垂直的長條型區域的顏色比其他區域的顏色還要深就會給 AC。

不過這樣並不會拿到滿分，而是會拿到一個介於 50 到 100 分之間的分數，要拿到滿分必須輸出一張「正確」的 1 才行，而分數就是依據你輸出的圖片和「正確」的圖片的 RMSE 誤差給分。或許會想到可以一次調整幾個像素試試看，但這樣大概要傳到天荒地老 (?) 才能解出正確的圖片。不過如果再回來看一次題目，會發現 Input Format 也是 1，事實上 input 就是一張 PNG 格式的「1」的圖片！剛好 TIOJ 的 Python 3 有 PIL，所以用 PIL 把輸入的圖片讀進來（不需要知道圖片是什麼格式，因為 PIL 會自動偵測），然後把 RGB 都 print 出來就可以拿到滿分了。

這題的提問系統也有提示：回覆是用 24×24 的圖片拼出來的。不過可能是圖片實在是長得太像文字了（事實上是直接截圖下來的，所以長得一模一樣），除非你的螢幕有超過 100% scaling 的設定會讓圖片會看起來有些模糊，不然肉眼除了 Yes. 有個可辨識的縮排以外是看不出差異的[5]，所以看起來也沒有人發現這個提示。

一些標程

pB. code、parse `registerBlock` 的 code
pD., pH. code
pE. 集中在同一筆測資做 error correction、直接生測資
pG. code
pH. Cado-NFS 的 config file、解密的 code
pI. 用來簽訊息的 code、生測資輸出的 code
pJ. code

[4] 如果你有看 2019 的 April Fools Contest，會發現那次的最後一題也是很類似的梗。
[5] 連出題者自己都被整到：在看到之前的回覆的時候還一度想說是不是忘記給這題的特殊回覆 xd