一中電研 37th 第二學期社內賽 題目解析

nevikw39

July 3, 2020

Abstract

1 旋轉矩陣 (rotate)

關鍵 GCD, LCM

1.1 題目

電電寫數學考卷遇到這麼一題：令一矩陣 A 表一平面上的線性變換 $\begin{pmatrix} \frac{-\sqrt{3}}{2} & \frac{-1}{2} \\ \frac{1}{2} & \frac{-\sqrt{3}}{2} \end{pmatrix}$，試問任一點 $P(x,y)$ 最少須經過多少次此變換後方回到原本的位置。

聰明的電電馬上想到 $A = \begin{pmatrix} \cos 150^\circ & -\sin 150^\circ \\ \sin 150^\circ & \cos 150^\circ \end{pmatrix} = R_{150^\circ}$ 即一個關於原點逆時針旋轉 150 度的變換，那麼只要旋轉成「一周角」的倍數就是旋轉一圈相當於沒有變換。

電電可以輕鬆地看出矩陣所代表的度數；因此，你的任務是幫電電計算最少旋轉幾次後回到原本的位置。

不過，電電國的角度單位有很多種，「一周角」並不總是 360 度。

1.1.1 輸入

兩個整數 a, b 皆 $< 2^{31}$，分別代表旋轉矩陣的度數及「一周角」的度數。

1.1.2 輸出

請輸出最少旋轉幾次後回到原本的位置。

1.2 解析

本題怕太水所以故意把題目敘述打很複雜，但是真的是考數學得到的靈感 XDD

很顯然本題所求為 $\frac{lcm(a,b)}{a} = \frac{\frac{a*b}{gcd(a,b)}}{a} = \frac{b}{gcd(a,b)}$。

C++ 可利用 <algorithm> 中為旋轉演算法提供之 `__gcd()` 函式。

1.3 參考程式碼

1.3.1 C++

```cpp
#include <bits/extc++.h>
using namespace std;
using namespace __gnu_pbds;
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int32_t a, b;
    cin >> a >> b;
    cout << b / __gcd(a, b) << '\n';
    return 0;
}
```

1.3.2 Python

```python
import sys
import math


def main():
    a, b = map(int, sys.stdin.buffer.readline().split
        ())
    sys.stdout.buffer.write(str(b // math.gcd(a, b)).
        encode())


if __name__ == "__main__":
    main()
```

2 割草終戰 (recall)

關鍵 條件

2.1 題目

由於 I4S◦ 菸粉綠蛆韓黑網軍蟑螂及國家機器動得非常厲害，我們先總統　蔣公在世、人類的救星、世界的偉人、自由的燈塔、民族的長城、宇宙的征服者，百年一見的政治鬼才 — 高譚市長韓總雞，遭遇地球文明史上最大的危機：罷免案。

依據《公職人員選舉罷免法》第 90 條：

> 罷免案投票結果，有效同意票數多於不同意票數，且同意票數達原選舉區選舉人總數四分之一以上，即為通過。有效罷免票數中，不同意票數多於同意票數或同意票數不足前項規定數額者，均為否決。

因為高譚市民非常非常多，有可能超過五百萬的五百萬倍，所以想請你幫忙寫個程式判斷罷免通過與否。

2.1.1 輸入

三整數 n, a, b 分別代表全體選舉人總數，有效同意、不同意票數。

$$0 \leq n < 2^{64}, a + b \leq n$$

2.1.2 輸出

首先請就罷免結果，輸出 `!!666` 或 `QQ` 。

接著請輸出「沉默不出來投票的韓粉」之人數。

2.2 解析

本題雖然就是個水題，但是有不少陷阱。

首先，由於 n 的範圍 2^{64} ，因此必須使用 `uint64_t`。其次，達到四分之一的門檻是 $\geq$ 而同意大於不同意則是 $>$，但須注意 C++ 的除法總是向零取整，假若寫成 `a >= n / 4` 則會存在誤差。另外，Python 使用 '/' 運算子的結果是浮點數，其誤差在 n 很大時誤差將很明顯。

因此，比較好的條件式是 `a * 4 <= n && a > b`，當然你若偏好位元運算可以寫成 `a << 2`。

2.3 參考程式碼

2.3.1 C

```c
#include <stdio.h>
#include <stdint.h>   // uint64_t
#include <inttypes.h> // SCNu64
int main()
{
    uint64_t n, a, b;
                                          // unsigned
       long long is far less portable.
    scanf("%" SCNu64 "%" SCNu64 "%" SCNu64, &n, &a, &b
       ); // you may use %l64u or %llu on different os
       , so it's better to use macro.
    puts(a << 2 >= n && a > b ? "!!666" : "QQ~~");
    printf("%" PRIu64, n - a - b);
    return 0;
}
```

2.3.2 Python

```python
import sys


def main():
```

```
    n, a, b = map(int, sys.stdin.buffer.readline().
        decode().split())
    sys.stdout.buffer.write(
        (("!!666\n%d" if a << 2 >= n and a > b else "
            QQ~~\n%d") % (n - a - b)).encode())


if __name__ == "__main__":
    main()
```

3 遞迴縮寫 (acronym)

關鍵 string

3.1 題目

所謂「遞迴縮寫」是指一組字串的縮寫恰好參照到他本身，例如 GNU Not Unix 的縮寫是 GNU ...

在資訊社群中，電神們有個慣例是傾向於使用「遞迴縮寫」的命名法來表達他們的幽默。

本題的任務是，請你檢查一組字串是否有可能是關於首字的「遞迴縮寫」。

3.1.1 輸入

一些以空白隔開的字串，數量及長度皆小於 25。

保證檔案結尾必定至少存在一個換行字元。

3.1.2 輸出

如果輸入的字串們可能存在遞迴縮寫，請輸出 o˙ó 否則 QQ。

3.2 解析

3.3 參考程式碼

3.3.1 C++

```
#include <bits/stdc++.h>
using namespace std;
inline char mytolower(const char &x) // convert a
    character to lower case by bitwise operation
{
    return x | 1 << 5;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    string s;
```

```
    cin >> s;
    for (int i = 1, l = s.length(); i < l; i++)
    {
        string t;
        cin >> t;
        if (mytolower(t[0]) != mytolower(s[i]))
        {
            cout << "QQ\n";
            return 0;
        }
    }
    getline(cin, s);
    cout << (s.empty() ? "o'_'o\n" : "QQ\n");
    return 0;
}
```

3.3.2 Python

4 二次曲線美化 (conic)

關鍵 迴圈、條件

4.1 題目

電電最近學到二次曲線，在電腦上打惹許多二次曲線的一般式 $(ax^2 \pm bxy \pm cy^2 \pm dx \pm ey \pm f; a, b, c, d, e, f \in \mathbb{N})$。可是寫的時候因為很趕很隨意，導致式子擠在一起很不方便閱讀，因此想請你寫個程式幫他美化重新排版。

排版的規則如下：

1. 各項必須依照降冪及字典順序出現
2. 指數應當出現在 *caret* 字元 '^' 後方，次數為一時不必表明次數
3. 常數項就只有常數
4. 唯具有非零係數的項可以出現，除非每項係數皆為 0，則常數項可以出現
5. 空格僅在二元運算子 $+$ —加— 和 $-$ —減— 的兩側出現
6. 若領導係數為正則毋需性質符號，反之應輸出一負號
7. 負項被視為減去正項，首項例外
8. 係數 ± 1 只有在常數項出現

4.1.1 輸入

輸入僅有一行，六個整數 a, b, c, d, e, f 以空白分隔，分別代表該二次曲線之各項。

4.1.2 輸出

輸出符合規則的二次多項式。

4.2 解析

本題是個比較複雜的條件判斷水題，沒有任何速解，就是需要耐心而已。

善用陣列可以大幅簡化程式碼。比如令一布林變數 `flag` 紀錄當前是否為首項，兩個字串陣列 ``plus[2] = {`` + '', ``''}``, ``minus[2] = {`` - '', ``-''}``，則在判斷性質符號只須輸出 `a[i] > 0 ? plus[flag] : minus[flag]`。

4.3 參考程式碼

4.3.1 C++

```cpp
#include <bits/stdc++.h>
using namespace std;
int inline myabs(int x) // absolute value function
    based on bitwise operation, which is a little bit
    efficient
{
    return (x ^ (x >> 31)) - (x >> 31);
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    const string plus[] = {"", " + "}, minus[] = {"-",
        " - "}, symbols[] = {"x^2", "xy", "y^2", "x",
       "y"};
    array<int, 6> a;
    for (int &i : a)
        cin >> i;
    bool flag = false;
    for (int i = 0; i < 5; i++)
        if (a[i])
        {
            cout << (a[i] > 0 ? plus[flag] : minus[
               flag]) << (myabs(a[i]) > 1 ? to_string(
               myabs(a[i])) : "") << symbols[i];
            flag = true;
        }
    if (a[5])
    {
        if (flag)
            cout << (a[5] > 0 ? " + " : " - ") <<
               myabs(a[5]);
        else
            cout << a[5];
    }
    else if (!flag)
        cout << '0';
    return 0;
}
```

4.3.2 Python

```python
import sys


def main():
    a = list(map(int, sys.stdin.buffer.readline().
        decode().split()))
    plus = ["", " + "]
    minus = ["-", " - "]
    symbols = ["x^2", "xy", "y^2", "x", "y"]
    flag = False
    for i in range(5):
        if a[i]:
            sys.stdout.buffer.write(((plus[flag] if a[
                i] > 0 else minus[flag]) +
                                     (str(abs(a[i]))
                                        if abs(a[i]) >
                                         1 else '') +
                                        symbols[i]).
                                        encode())
            flag = True
    if a[5]:
        if flag:
            sys.stdout.buffer.write(
                ((" + %d" if a[5] > 0 else " - %d") %
                    abs(a[5])).encode())
        else:
            sys.stdout.buffer.write(str(a[5]).encode()
                )
    elif not flag:
        sys.stdout.buffer.write('0'.encode())


if __name__ == "__main__":
    main()
```

5 數字朗讀 (num)

關鍵 迴圈、遞迴

5.1 題目

電電欲令電腦朗讀出數字，他現在已經有英文轉語音 *(TTS)* 的 **API** 惹。

在英文中與中文不同，差兩次以上的位數並不需要「零」或 “end” （比賽結束後你可以請 Google 小姐念給你聽）。此外，正式的寫法如支票，並不會用 “a hundred” 等等而應該是 “one hundred”。

其他英文數字讀法規則都跟你小時候一樣（吧

5.1.1 輸入

僅有一整數 $n < 2^{31}$。

5.1.2 輸出

n 在英文的念法。

5.2 解析

本題分為兩個子題。第一小題保證 $0 \le n < 1000$，我們可以嘗試利用遞迴解之。定義函式：

$$num\_to\_str(n) = \begin{cases} digits[n], & n < 20 \\ tys[n/10] + \text{`` }-\text{''} + num\_to\_str(n\%10), & n < 100 \\ digits[n/100] + \text{``}hundred\text{''} + num\_to\_str(x\%100), & n < 1000 \end{cases}$$

其中，`digits[]` 表小於二十之非負整數之英文，`tys[]` 表各「十」之英文。當然，上揭定義過於簡化，實際上還須考慮遞迴下去若為空則不應該輸出，且本題為嚴格比對，須特別留意。

第二小題 $-2^{31} \le n < 2^{31}$，首先面對負數可以輸出 `minus` 並轉為正數處理，接下來每三位一小節，若 $n < 10^{12}$ 則輸出 $num\_to\_str(n/10^9)$ + " billion" + $num\_to\_str(n\%10^9)$，以此類推。

5.3 參考程式碼

5.3.1 C++

```
#include <bits/stdc++.h>
using namespace std;
const string num_to_words(int x)
{
    static const string digits[] = {"", "one", "two",
        "three", "four", "five", "six", "seven", "eight
        ", "nine", "ten", "eleven", "twelve", "thirteen
        ", "fourteen", "fifteen", "sixteen", "seventeen
        ", "eighteen", "nineteen"},
                        tys[] = {"", "", "twenty", "
                            thirty", "forty", "fifty",
                            "sixty", "seventy", "eighty
                            ", "ninety"};
    if (x < 0)
        return "minus " + num_to_words(-x);
    if (x == 0)
        return "zero";
    if (x < 20)
        return digits[x];
    if (x < 100)
        return tys[x / 10] + ((x % 10) ? '-' +
            num_to_words(x % 10) : "");
    if (x < 1000)
```

```
        return digits[x / 100] + " hundred" + ((x %
            100) ? ' ' + num_to_words(x % 100) : "");
    if (x < 1000000)
        return num_to_words(x / 1000) + " thousand" +
            ((x % 1000) ? ' ' + num_to_words(x % 1000)
            : "");
    if (x < 1000000000)
        return num_to_words(x / 1000000) + " million"
            + ((x % 1000000) ? ' ' + num_to_words(x %
            1000000) : "");
    if (x < 1000000000000)
        return num_to_words(x / 1000000000) + "
            billion" + ((x % 1000000000) ? ' ' +
            num_to_words(x % 1000000000) : "");
    return "error";
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    cout << num_to_words(n) << '\n';
    return 0;
}
```

5.3.2 Python

```
import sys


def num_to_words(x: int):
    digits = ["", "one", "two", "three", "four", "five
        ", "six", "seven", "eight", "nine", "ten", "
        eleven",
               "twelve", "thirteen", "fourteen", "
                   fifteen", "sixteen", "seventeen", "
                   eighteen", "nineteen"]
    tys = ["", "", "twenty", "thirty", "forty",
           "fifty", "sixty", "seventy", "eighty", "
               ninety"]
    if x < 20:
        return digits[x]
    if x < 100:
        return tys[x // 10] + (('-' + digits[x % 10])
            if x % 10 else '')
    if x < 1000:
        return digits[x // 100] + " hundred" + ((' ' +
             num_to_words(x % 100)) if x % 100 else '')
```

```
def main():
    n = int(sys.stdin.buffer.readline())
    if not n:
        sys.stdout.buffer.write("zero".encode())
    if n < 0:
        sys.stdout.buffer.write("minus ".encode())
        n = -n
    if n >= 1000000000:
        sys.stdout.buffer.write(
            (num_to_words(n // 1000000000) + " billion
                ").encode())
        n %= 1000000000
        if n:
            sys.stdout.buffer.write(' '.encode())
    if n >= 1000000:
        sys.stdout.buffer.write(
            (num_to_words(n // 1000000) + " million").
                encode())
        n %= 1000000
        if n:
            sys.stdout.buffer.write(' '.encode())
    if n >= 1000:
        sys.stdout.buffer.write(
            (num_to_words(n // 1000) + " thousand").
                encode())
        n %= 1000
        if n:
            sys.stdout.buffer.write(' '.encode())
    sys.stdout.buffer.write(num_to_words(n).encode())


if __name__ == "__main__":
    main()
```

6 費事級數 (Fibonacci)

關鍵 數論 Matrix

6.1 題目

電電觀察兔子族群的個數，發現一開始有一對兔子，兩個單位時間後她們就可以繁殖出一對兔子；兔子永不死去。據此，我們可以歸納出兔子對數：

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

你以為這題叫你算截至 n 單位時間前有幾對兔子ㄇ？？錯錯錯！！電電感興趣的是兔子所消耗之飼料的總和，其中每對兔子每單位時間須消耗一單位之飼料，所求即 $\sum_{i=0}^{n-1} F_i$。

兔子的個數由於很難算，因而被命名為「費事數列」，而這個數列之和自然就是「費事級數」惹。

因為答案顯然可能很大，所以請對 $10^9 + 7$ 取模。

6.1.1 輸入

一非負整數 $0 \leq n < 2^{31}$。

6.1.2 輸出

請輸出 $\sum_{i=0}^{n-1} F_i$。

6.2 解析

本題暗示有夠明顯，就是要求費氏數列的和。

6.2.1 遞迴：$O(2^n)$ Bad!!

計算費氏數列最入門的方法，遞迴，有許多改進的空間，例如求 `F(n)` 而呼叫 `F(n-1)` 及 `F(n-2)`，復各自呼叫 `F(n-2)`、`F(n-3)`、`F(n-3)` 及 `F(n-4)`，在過程中會不斷重複呼叫同樣的值，十分浪費。簡單證明時間複雜度：$T(n) = T(n-1) + T(n-2) \in O(F(n))$ 也是費氏數列，不過通常也可以寫成 $O(2^n)$，因為費氏數量各項之比趨近於黃金比例 $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ 而 $Big - O$ 表緊的漸進上界，故 $O(2^n)$ 合理。

其實，那些在教遞迴時提到費氏數列或在教費氏數列時提到遞迴的人實在是誤人子弟。費氏數列確實是遞迴數列，但不代表要以遞迴計算之啊！！正常人如何數費氏數列？？$1, 1, 2, 3, 5, 8, \cdots$ 一個一個加啊！！假若你看到費氏數列只想到遞迴，那你中毒太深惹，可憐哪。

6.2.2 遞推：$O(n)$

離題惹。對於遞迴常見的加速手段有記憶化搜索 *(Memoization)*（是的，很有動態規劃 *(Dynamic Programming, DP)* 的味道），也就是說，在每次計算一個 $F(n)$ 時，就把他存入陣列，下次就別再算一遍，這是一個由上而下 *(top-down)* 的方法。另外，更人類的作法也可以從 $F(0) = 0, F(1) = 1$ 開始由下而上 *(bottom-up)* 遞推/迭代 *(iterate)*。*bottom-up* 的好處是可以順便計算和。顯然兩者的時間複雜度皆為 $O(n)$。

取模的部份要小心。費氏數列增長很快，有關數論的題目通常會要求對一個很大的質數取模。取模是為惹防止整數溢出，畢竟沒有必要刁難大數。因此，在每個加乘法運算的過程中都應當取模，否則還是有可能溢出，那麼答案就錯誤惹。

附帶一提，上述之所以成立，是由於 $(A + B)\%M = (A\%M + B\%M)\%M$ 且 $(A * B)\%M = (A\%M * B\%M)\%M$。然而，模運算中只有除法沒有分配律，所以才需要所謂「關於模運算的乘法反元素」，簡稱模逆元。

6.2.3 矩陣：$O(\log n)$

又再度離題惹。完成上述操作，你最多還是會得到 **NA 80%**。剩下四筆測資該如何通過呢？？此時，我們需要數學的幫忙惹。不知道你是否有印象，徐氏數學第四冊第三章矩陣課後練習或精彩命題圈有遞迴數列與矩陣之結合？？沒錯就是他！！

首先令 $S_n = \sum_{i=0}^{n} F_i$，依據定義可得：

$$\begin{array}{rlllll} F_1 = & 1 & & \\ F_2 = & 2 & & \\ F_3 = & F_2 & + & F_1 \\ F_4 = & F_3 & + & F_2 \\ F_5 = & F_4 & + & F_3 \\ \vdots & \vdots & & \vdots \\ +) \quad F_n = & F_{n-1} & + & F_{n-2} \\ \hline S_n = & (S_{n-1} - F_1) + & & S_{n-2} + 2 \end{array}$$

列出矩陣轉移式：

$$\begin{pmatrix} S_n \\ S_{n-1} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} S_{n-1} \\ S_{n-2} \\ 1 \end{pmatrix}$$

接著利用矩陣快速冪，即可在 $O(\log n)$ 內算出 S_n。

最後最後，本題還有更簡單的矩陣解法。對於費氏數列，我們有此性質：$\sum_{i=1}^{n} F_i = F_{n+2} - 1$。也就是說，二階矩陣快速冪求 $F_{n+2} - 1$ 即可。

費氏數列實在太漂亮惹，還有好多好多性質可以玩呢。

https://blog.yangjerry.tw/2019/01/31/fibonacci-is-bigO1/

6.3 參考程式碼

6.3.1 C++

```cpp
#include <bits/stdc++.h>
using namespace std;
template <typename T = int64_t, int N = 2, int M = int
    (1e9 + 7)>
struct matrix
{
    using vec = array<T, N>;
    array<vec, N> a{};
    matrix() = default;
    matrix(const matrix &x) = default;
    matrix(const initializer_list<T> &il)
    {
        assert(il.size() == N * N);
```

```
        auto itr = il.begin();
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                a[i][j] = *itr++;
    }
    matrix operator*(const matrix &x) const
    {
        matrix y;
        for (int k = 0; k < N; k++)
            for (int i = 0; i < N; i++)
                for (int j = 0; j < N; j++)
                    y.a[i][j] = (y.a[i][j] + a[i][k] *
                        x.a[k][j] % M) % M;
        return move(y);
    }
    vec operator*(const vec &x) const
    {
        vec y{};
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                y[i] = (y[i] + a[i][j] * x[j] % M) % M
                    ;
        return move(y);
    }
    matrix power(T n) const
    {
        matrix y, x = *this;
        for (int i = 0; i < N; i++)
            y.a[i][i] = 1;
        while (n)
        {
            if (n & 1)
                y = y * x;
            x = x * x;
            n >>= 1;
        }
        return move(y);
    }
};
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    matrix<> base{1, 1, 1, 0};
    cout << (base.power(n + 2UL) * array<int64_t,
       2>{0, 1})[0] - 1 << '\n';
    return 0;
}
```

7 摯友圈圈

關鍵 Graph DFS

7.1 題目

如果有一群人兩兩之間都是摯友，那麼我們稱他們是一個摯友圈圈。

現在有編號 0 $N-1$ 的 N 個人，以及他們之間的摯友關係，請輸出最大摯友圈圈的人數。

7.1.1 輸入

第一行有兩個整數 $N, M(0 < N \leq 22, 0 \leq M < 69)$ ，分別代表有 N 個人和 M 筆摯友關係。

接下來有 M 行，每行有兩個整數 $a, b(0\ a, b\ N)$ 表示 a 和 b 互為摯友。

7.1.2 輸出

最大摯友圈圈的人數。

7.2 解析

簡單的 **DFS** 問題，從每一點開始暴搜即可。

BFS 應該也行。

7.3 參考程式碼

7.3.1 C++

```
#include <bits/extc++.h>
using namespace std;
using namespace __gnu_pbds;
int n, mx = 0;
vector<gp_hash_table<int, null_type>> g;
vector<int> p;
void dfs(int x)
{
    mx = max(mx, (int)p.size());
    for (const int &i : g[x])
    {
        bool flag = true;
        for (const int &j : p)
            if (g[i].find(j) == g[i].end())
                flag = false;
        if (!flag)
            continue;
        p.push_back(i);
```

```
        dfs(i);
        p.pop_back();
    }
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int m;
    cin >> n >> m;
    g.resize(n);
    while (m--)
    {
        int a, b;
        cin >> a >> b;
        g[a].insert(b);
        g[b].insert(a);
    }
    for (int i = 0; i < n; i++)
        dfs(i);
    cout << (mx ?: 1);
    return 0;
}
```