演算法中的線性代數

詹挹辰

August 31, 2021

目錄

0 前言 **2**
0.1 內容大綱 . . . 2
0.2 動機 . . . 2
0.3 心得 . . . 2

1 一些定義 **3**
1.1 線性代數 . . . 3
1.2 矩陣 . . . 3
1.3 向量 . . . 3
1.4 Vector Space . . . 4

2 矩陣 **5**
2.1 矩陣乘法 . . . 5
2.1.1 Strassen Algorithm . . . 5
2.2 加速線性遞迴 . . . 6
2.2.1 矩陣快速冪 . . . 6
2.2.2 補充：求一般式 . . . 7
2.3 鄰接矩陣 . . . 7
2.3.1 定長路徑統計 . . . 7
2.3.2 定長最短路 . . . 7
2.4 矩陣的結合律 . . . 8
2.5 其他 . . . 8

3 高斯消元 **8**
3.1 Gauss Jordan Elimination . . . 8
3.2 求解線性方程組 . . . 9
3.3 反矩陣 . . . 10
3.4 基底 . . . 10
3.5 行列式 . . . 10
3.5.1 高斯消元求行列式 . . . 10
3.5.2 行列式應用 . . . 11

4 線性基 **12**
4.1 線性基介紹 12
4.2 建構 13
4.3 應用 14

5 矩陣樹定理 **15**
5.1 無向圖生成樹計數 15
5.2 有向圖生成樹計數 15
5.3 帶權生成樹計算 16
5.4 整理 16

6 參考文獻 **16**

0 前言

0.1 內容大綱

本文會以高中所學到的線性代數，也就是向量、矩陣、以及高斯消元作爲基礎進行延伸。本文將會引入一些的線性代數的知識，並提出這些知識在演算法競賽中常見的用途。

0.2 動機

這學期的數學課程中學到了矩陣，矩陣在數學中是相當重要的一個領域，特別是在線性代數。而線性代數在現今也是許多演算法中的基礎。剛好演算法競賽是我的強項，而線性代數也是相當重要的領域。因此決定趁著此次機會延伸學習，並且連結矩陣、線性代數在演算法競賽中的應用。

0.3 心得

線性代數在演算法競賽中出現的次數雖然不多，但我認爲線性代數能解決的問題非常的有趣。編寫這份筆記讓我學習到了很多有趣的線性代數的用處，也讓我重新審視自己之前的所學。在編寫這份筆記的過程中最令我感到驚艷的是高斯消去法除了單純解決聯立方程組之外還有如此多不同的用途。而線性基加速高斯消元的這個想法也令我感到相當新奇。

1 一些定義

由於這份筆記主要探討的是演算法中的線性代數，故這邊會省略掉高中有教過的定義。這些定義在之後的部分會用到。

1.1 線性代數

線性代數是什麼？讓我們來看一下維基百科的定義：線性代數是關於向量空間和線性映射的一個數學分支。它包括對線、面和子空間的研究，同時也涉及到所有的向量空間的一般性質。

基本上可以說是在向量空間中處理有關於矩陣和向量的線性變換的各種內容。而這份筆記會提到一些向量空間觀念以及各式矩陣相關演算法。需要注意的是這邊的" 線性" 指的並非是" 函數圖像爲一直線"。之所以會叫線性是因爲前人在命名時線性代數還未完善，因而有此命名。

1.2 矩陣

高中的定義省略，這邊定義兩個下文會用到的名詞。

Definition. 簡化列階梯形矩陣 (reduced row echelon form)

1. 所有非零列在所有全零列的上面。即全零列都在矩陣的底部。
2. 非零列的首項係數（leading coefficient），也稱作主元，即最左邊的首個非零元素，嚴格地比上面列的首項係數更靠右。
3. 每個首項係數是 1，且是其所在行的唯一的非零元素。

若一個矩陣滿足上述三個性質，則稱之爲簡化列階梯形矩陣。

Definition. 矩陣的秩 (rank)
定義矩陣 A 的秩 (rank) 爲線性獨立的橫行的最大數目，也就是將 A 化爲簡化列階梯形矩陣後非零列的數量，記爲 $rank(A)$。

其中線性獨立會在 vector space 的部分介紹。

1.3 向量

Vector 通常在線性代數我們會討論的是 column vector，也就是直著寫的向量。而 column vector 其實就是一個 $n \times 1$ 的矩陣。爲了不占版面，有時我們會將 column vector 寫成轉置矩陣的樣子。另外，我們定義 0 向量的表示符號爲 0。最後我們有以下定義：

Definition. $\mathbb{R}^n$ 代表所有由 n 個實數來構成的 column vector 的集合，也就是所有 $n \times 1$ 的實數矩陣的集合。

1.4 Vector Space

在講 Vector Space 前我們需要先知道體 (Field) 這個東西：

Definition. (體 Field) 體 $\mathbb{F}$ 是一個集合，他具有加法和乘法 (係數積) 兩種運算，並且具有交換率、單位元、以及反元素 (0 沒有乘法反元素)。而常見的體有實數 $\mathbb{R}$ 或是模 p 底下的運算 F_p。

礙於篇幅限制這裡就不明確定義體這個東西，我們只要知道他是一個滿足上面限制的集合就好了。接下來我們來一個抽象的定義：

Definition. (Vector Space) 一個向量空間 V 是一個由許多向量與一個可以作用在向量的係數上的一個體 $\mathbb{F}$ 構成的集合，這些向量滿足

1. 向量加法具封閉性且具有結合率，交換率。也就是說如果 $u, v, w \in V$，則 $u + v \in V$，$(u + v) + w = u + (v + w)$ 以及 $u + v = v + u$。
2. 存在零向量 (加法單位元素)0 使得 $v + 0 = v, \forall v \in V$。
3. 對於每個向量 v 存在反元素 $-v$ 滿足 $v + (-v) = 0$。
4. 係數積具封閉性並且有結合率和加法分配率，也就是如果 $a, b \in \mathbb{F}$，$u, v \in V$，則 $au \in V$，$a(bv) = (ab)v$，$(a + b)v = av + bv$，$a(u + v) = au + av$。
5. 存在乘法單位元素 $1 \in F$ 滿足 $1v = v, \forall v \in V$。

常見的向量空間有 $\mathbb{R}^n$ 以及模 2 下的向量 $\mathbb{F}_2^n$。($\mathbb{F}_2^n$ 也就是 n-bit 的整數，其中加法定義爲 xor 操作的向量空間)。

值得一提的是向量空間他的定義是抽象的，我們可能不會知道他裡面長怎樣，我們只知道他有著上面那樣的限制。即便如此，我們實務上比較常用到的通常會是我們所能理解的東西 ($\mathbb{R}^n$ 之類的)。

接著來定義更多的東西：

Definition. (子空間 Subspace) 如果一個向量空間的子集 W 滿足在裡面的向量加法以及係數積都封閉，且 $0 \in W$，我們就說 W 是一個子空間 (subspace)。

Definition. (線性獨立 linear independent) 對於向量 $v_1, v_2, ..., v_n$，如果 $a_1v_1 + a2_v2 + \cdots + a_nv_n = 0 \Rightarrow a_i = 0 \forall i$，也就是任何向量都不能用其他的向量湊出來，那麼我們就說這些向量線性獨立。

Definition. (生成集合 spanning set) 對於向量 $v_1, v_2, ..., v_n$，如果一個向量空間 V 的所有向量 v 都可以被寫成 $v = a_1v_1 + a_2v_2 + \cdots + a_nv_n$，那麼我們就說向量 $v_1, v_2, ..., v_n$ 是 V 的一個生成集合。額外定義集合 $S = \{v_1, v_2, ..., v_n\}$ 所能生成的所有向量的集合叫做 $span(S)$。此時有 $V \subseteq span(S)$。

Definition. (基底 basis) 對於向量 $v_1, v_2, ..., v_n$ 跟一個向量空間 V，如果這些向量線性獨立並且生成 V，那麼這些向量就是 V 的一個基底。而有一個定理是這樣的：每個向量空間的任意一個基底的大小都是一樣的。而該向量空間的維度被定義爲其基底的大小。

2 矩陣

這邊介紹一些矩陣相關的有趣演算法。接下來如果沒有特別說明，都考慮矩陣是 $n \times n$ 的方陣。而 A 是一個 $n \times n$ 的方陣。

2.1 矩陣乘法

在高中數學當中我們有學過矩陣的乘法，如果按照高中的定義來做乘法的話，時間複雜度會是 $O(N^3)$ 的。而事實上有一個常用的矩陣乘法可以做到 $O(N^{\log_2 7}) \approx O(N^{2.801})$：Strassen Algorithm。

2.1.1 Strassen Algorithm

Strassen Algorithm 是一個基於分治的作法。先假設我們有兩個大小是 $2^k \times 2^k$ 的矩陣 A, B，並且令 $C = AB$。我們將這三個矩陣分成三個 2×2 的矩陣，也就是

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

其中 C_{11} 代表 C 這個矩陣左上角的 $2^{k-1} \times 2^{k-1}$ 矩陣，其他位置以此類推。按照上面的分法，乘法規則變成 $C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j}$。若使用傳統的矩陣乘法，這樣會需要 8 次乘法運算以及 4 次加法運算。而 Strassen 則只需要 7 次乘法運算以及 18 次加法運算 (加法複雜度 $O(N^2)$)。作法如下：

$$
\begin{aligned}
P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
P_2 &= (A_{21} + A_{22})B_{11} \\
P_3 &= A_{11}(B_{12} - B_{22}) \\
P_4 &= A_{22}(B_{21} - B_{11}) \\
P_5 &= (A_{11} + A_{12})B_{22} \\
P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
P_7 &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned}
$$

$$
\begin{aligned}
C_{11} &= P_1 + P_4 - P_5 + P_7 \\
C_{12} &= P_3 + P_5 \\
C_{21} &= P_2 + P_4 \\
C_{22} &= P_1 + P_3 - P_2 + P_6
\end{aligned}
$$

以上作法可以透過展開得到證明。至於複雜度的部分，令當下爲 $N \times N$ 矩陣相乘：

$$T(N) = 7T(\frac{N}{2}) + O(18N^2)$$

由主定理 (master theorem) 可以得知這個遞迴式的複雜度是 $O(N^{\log_2 7})$ 的。如果矩陣不是二的冪次的方陣，不妨把他們補到二的冪次。詳細來講就是補 0 補到大小是二的冪次。

雖然這個演算法在競賽中不常用到 (數字不夠大)，但是在實務上 Strassen 會是一個提升效率不錯的作法。

2.2 加速線性遞迴

2.2.1 矩陣快速冪

這個可以說是矩陣在演算法競賽的應用中最常見的一個了。來回顧一個經典問題：Fibonacci 數列，也就是 $F_0 = F_1 = 1, F_i = F_{i-1} + F_{i-2}, \forall i \geq 2$ 這個問題已經在演算法競賽界中相當有名了，如果我們把 F_{i-1}, F_{i-2} 當作一個 column vector，那麼我們可以透過矩陣 A 進行轉移：

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{i-1} \\ F_{i-2} \end{bmatrix} = \begin{bmatrix} F_i \\ F_{i-1} \end{bmatrix}$$

由此作法可以推得 $[F_n, F_n - 1]^T = A^{n-1}[F_1, F_0]^T$，於是我們可以在 $O(2^3 logN)$ 的時間求得 F_n。

只要同樣是線性遞迴都可以用這個方式，複雜度爲 $O(K^3 logN)$，其中 K^3 爲矩陣乘法。特別的是如果線性遞迴裡面有單獨出現 n^k 項的也可以 (例如：$F_i = F_{i-1} + F_{i-2} + n$)。至於矩陣的構造就留給大家自行思考。

2.2.2 補充：求一般式

這邊有一個與演算法競賽沒什麼關連但是相當有趣的資訊。上文中的 F_n 一般式其實可以由線性代數求得。但是這個做法需要用到矩陣的 eigenvalue(特徵值) 以及 eigenvector(特徵向量) 來求解。其中作法大概是先求出該矩陣的 eigenvalue, eigenvector，然後透過相似轉換以及對角化。如此一來即可快速得到冪次的答案。也就可以推出一般式了。但由於這個作法與演算法競賽無關，有興趣的人可以自行上網搜尋資料，這裡不多加說明。

2.3 鄰接矩陣

鄰接矩陣在圖論中常常可以告訴我們很多訊息。

2.3.1 定長路徑統計

給你一張 N 個點的圖，求解所有點對 (u, v) 從 u 走到 v 經過 K 條邊的路徑的數量。

用鄰接矩陣 G 來表示這張圖，其中如果 u, v 之間有邊那麼 $G_{u,v} = 1$，反之則爲 0。令已經走過 k 條邊的答案爲 C_k，那麼根據動態規劃，我們有

$$C_{k+1\ (i,j)} = \sum_{p=1}^{n} C_{k\ (i,p)} \cdot G_{p,j}$$

而這個過程其實也就是在做矩陣乘法，於是我們有 $C_k = G^k$。所以我們就得到了一個複雜度 $O(N^3 logK)$ 的作法了。

2.3.2 定長最短路

給你一張 N 個點的帶邊權圖，求解所有點對 (u, v) 從 u 走到 v 經過 K 條邊的最短路。

用鄰接矩陣 G 來表示這張圖，其中如果 u, v 之間有邊那麼 $G_{u,v}$ 爲其邊權，反之則爲無限大。令已經走過 k 條邊的答案爲 C_k，一樣考慮使用動態規劃，我們有：

$$C_{k+1\ (i,j)} = \min_{1 \le p \le N} \{C_{k\ (i,p)} + G_{p,j}\}$$

雖然我們沒辦法直接對他進行矩陣乘法，但是我們可以對他進行類似矩陣乘法的操作。定義這個運算爲 $\odot$，那麼我們有 $C_{k+1} = C_k \odot G$。而 $\odot$ 操作具有結合率，所以可以推得 $C_k = G^{\odot k}$。那麼我們也就可以使用矩陣快速冪得到 $O(N^3 logK)$ 的作法。

2.4 矩陣的結合律

在做動態規劃類型的題目的時候，有時候題目會帶修改，也就是所謂的動態 DP。而在 2.2 的地方我們有提到，如果 DP 式是線性遞迴的形式，我們可以將轉移的過程視爲矩陣乘法。而此時如果有修改操作的話，由於矩陣乘法有著結合律這件事情，也因此我們可以透過線段樹來維護矩陣！

2.5 其他

矩陣還有許多有趣的功用，例如 eigenvalue 的應用等等，如果有興趣的讀者可以自行上網搜尋。

3 高斯消元

在高中的時候我們就有學過高斯消元了，而高中的高斯消元主要用途在於求解線性方程組。但高斯消元能做的事不僅僅是求出線性方程組。接下來我們要來介紹幾個常見的應用。進入高斯消元前，我們先定義：

3.1 Gauss Jordan Elimination

高斯消元即透過三種基礎列運算來想辦法把矩陣變成一個上三角矩陣的過程。高斯消元有一個延伸的做法叫做 Gauss Jordan Elimination，這個做法是將結果變成對角矩陣 (簡化列階梯形矩陣) 的作法。而普通的高斯消元只會把他變成上對角矩陣。這邊我們介紹高中沒有提到的 Gauss Jordan Elimination.

Gauss Jordan 跟標準的高斯消元並沒有太大的差異，都是依序由左至右處理每一行，第 i 行的處理方式大致如下：

1. 從還沒被用過的列找到第 i 行一個最大的非 0 的元素，並且把該元素的那列跟第 i 列交換
2. 將 $A_{i,i}$ 變成 1
3. 透過加減消去把第 i 行的其他元素都消成 0

結果出來會是一個列階梯形矩陣。但由於某些矩陣不一定可以被消成對角矩陣，也就是某行全爲 0 的時候。所以根據題目特性高斯消元出來的結果可能會不是對角矩陣。

這裡提供參考程式碼：

```cpp
void gauss(vector<vector<double>> &v) {
    // 大小爲 n * m 的矩陣, 採用 0-base
    // 解線性方程組用
    const double eps = 1e-9;
    if(v.empty()) return;
    int n = v.size(), m = v[0].size();
    int line = 0;
    for(int i = 0; i < m; i ++) {
        int p = -1;
        for(int j = line; j < n; j ++) {
            if(fabs(v[j][i]) < eps) continue;
            if(p == -1 or fabs(v[j][i]) > fabs(v[p][i])) {
                p = j;
            }
        }
        if(p == -1) continue;
        for(int j = 0; j < m; j ++) {
            swap(v[line][j], v[p][j]);
        }
        double tmp = v[line][i];
        for(int j = 0; j < m; j ++) {
            v[line][j] /= tmp;
        }
        for(int j = 0; j < n; j ++) {
            if(j == line) continue;
            double t = v[j][i] / v[line][i];
            for(int k = 0; k < m; k ++) {
                v[j][k] -= t * v[line][k];
            }
        }
        line ++;
    }
}
```

特別的，正常的高斯消元都是在實數或是整數下運作。而如果所有操作都是在模 2 底下進行的話，可以使用位元運算加速運算，複雜度會變成 $O(\frac{n^3}{w})$，w 根據資料型態而定。(xor 是模 2 底下的加法運算)

至於高斯消元可以做什麼呢？我們接下來來介紹幾個常見應用

3.2 求解線性方程組

這也就是我們高中所學到的應用，也就是求解多元一次方程。作法就是寫成增廣矩陣後進行高斯消元。由於高中課程已經有介紹過了，這邊就不再多加說明。

3.3 反矩陣

反矩陣 A^{-1} 也可以由高斯消元求出。作法爲在 A 的右邊放一個單位矩陣 I 變成一個新的矩陣，也就是 $[A \mid I]$。將這個矩陣進行高斯消元，把左半邊消成 I，而右半邊就會是 A^{-1}。換句話說也就是消完之後矩陣就會變成 $[I \mid A^{-1}]$。

至於證明的部分，因爲每個基礎列運算可以等價於進行一次矩陣乘法。所以如果對 A 進行若干次的基礎列運算使其變成 I 的話，也就代表存在一系列的矩陣 $E_1, E_2 \dots E_t$ 使得 $E_1 E_2 \dots E_t A = I$。那麼 $E_1 E_2 \dots E_t$ 就會是 A 的反矩陣。

特別的，若 $rank(A) \neq n$，那麼則不存在反矩陣。也就是說，如果高斯消元完的結果如果左邊不是 I 的話則不存在反矩陣。

而反矩陣有什麼功用呢？反矩陣主要用於求解形如 $Ax = b$ 的方程式。此時僅需將 A^{-1} 作用在左右兩式即可得到 $x = A^{-1}b$。

3.4 基底

透過高斯消元也可以求出 $v_1, v_2 \dots v_n$ 的基底。高斯消元完後剩下多少系數不全爲 0 的向量即爲基底的大小。證明可以從線性獨立定義得到。雖然基底貌似現在看來沒有多大用處，但事實上基底有很多可以發揮它的功用的地方。在章節 4 我們會介紹更多有關基底的用途。

3.5 行列式

3.5.1 高斯消元求行列式

在高中的時候我們只有學到二階以及三階的行列式求法。而行列式其實有著拓展到 n 階的公式：

$$\det A = \sum_{p} (-1)^{\tau(p)} \prod_{i=1}^{n} A_{i,p_i}$$

其中 p 是一個 $1 \sim n$ 的排列，而 $\tau(p)$ 代表 p 中的逆序對數。如果直接按照定義來計算的話整體複雜度會是 $O(n! \cdot n)$，相當不可愛。但是利用高斯消元我們就可以把他做到 $O(n^3)$ 的複雜度。

在高中的時候我們有學過一些關於行列式的性質：

1. 交換兩列行列式值乘以-1
2. 同一列可以提出一個常數 k 乘在外面
3. 任一列加上任一其他列的常數倍其行列式值不變

我們可以發現這三項其實可以對應到高斯消元的三種基本列運算，所以我們就可以把矩陣 A 先使用高斯消元把 A 消成上三角矩陣之後再去計算其行列式值。由於上三角矩陣的行列式值即爲對角線所有數的乘積，於是就可以在整體 $O(n^3)$ 的複雜度下算出 $det(A)$。

特別的，因爲行列式值出來通常會很大，所以運算過程通常會需要對 P 進行模運算。而若模數不是質數的話逆元不一定存在，所以無法進行除法操作。此時可以使用輾轉相除法進行加減消去的動作。而由於輾轉相除法的過程每次操作會使 $A_{i,j}$ 減少至少一半，所以均攤下來輾轉相除的複雜度會是 $O(n^2 \log P)$，故總複雜度爲 $O(n^2(n + \log P))$。

3.5.2 行列式應用

行列式在線性代數和實務上是非常重要的一個工具，但是他在演算法競賽中出現的頻率可以說是少之又少。即便如此，行列式還是會有可能會出現在演算法競賽，而行列式出現的話有高機率是從他的定義下手。讓我們來看一道例題。

Problem 3.1. Codeforces 736D
給定一張兩邊各有 n 個點，m 條邊無重邊的二分圖。已知此二分圖不同的完美匹配方法有奇數個。現在對於每條邊，請輸出拔掉該條邊後完美匹配的方法數是奇數個還是偶數個。$n \leq 2000, m \leq 10^5$

將這個二分圖用鄰接矩陣 A 表示，並且令兩個點集爲 X, Y，$X = x_1, x_2, \ldots, x_n, Y = y_1, y_2, \ldots, y_n$。此時我們枚舉所有的點對配對會不會形成一個完美匹配，也就是 $\prod A_{i,p_i} = 1$ 時爲一個完美匹配，其中 p 是一個 $1 \sim n$ 的排列。於是完美匹配的數量就會是：$\sum_p \prod A_{i,p_i}$。而我們可以發現，由於運算是在模 2 底下進行，所以

$$\det A = \sum_p (-1)^{\tau(p)} \prod_{i=1}^{n} A_{i,p_i} \equiv \sum_p \prod A_{i,p_i} \pmod 2$$

因此 $\det A \pmod 2$ 就是完美匹配方法數的奇偶性。

至於拔掉邊 (x_i, y_j) 的答案也就是將 $A_{i,j}$ 設爲 0 後行列式的值。但由於我們不可能暴力做出 m 次的行列式值，所以我們要引入一些定義跟公式：

Definition. (子行列式、餘因子、伴隨矩陣)

- 子行列式 (minor)：對一個 $n \times n$ 的矩陣 A，令在 (i, j) 的子行列式 $M_{i,j}$ 爲刪掉 A 的第 i 列和弟 j 行所得到的矩陣的行列式的值。
- 餘因子 (cofactor)：令 $C_{ij} \coloneqq (-1)^{i+j} M_{ij}$，稱爲 A 在 (i, j) 的餘因子。

- 伴隨矩陣 (adjugate matrix)：定義 $\mathrm{adj}(A) \coloneqq [C_{i,j}]^T$，其中 $\mathrm{adj}(A)$ 稱作 A 的伴隨矩陣

那麼我們會有：

Theorem 3.1. $A\mathrm{adj}(A) = (\det A)I$，特別的，如果 A^{-1} 存在，那麼 $\mathrm{adj}(A) = (\det A)A^{-1}$。

而上面這個定理則可以告訴我們這件事：

Theorem 3.2. 行列式降階 (拉普拉斯展開)

$$\det A = ((\det A)I)_{i,i} = \sum_{j=1}^{n} A_{i,j}\mathrm{adj}(A)_{j,i} = \sum_{j=1}^{n} A_{i,j}(-1)^{i+j}M_{i,j}$$

那麼我們就可以得到以下兩件事：

1. 因爲 $\det A \equiv 1 \pmod 2$，由定理 3.1 可以知道 $\mathrm{adj}(A) = A^{-1}$
2. 由行列式降階可知，若將 $A_{i,j}$ 變成 0，那麼行列式值會減少 $M_{i,j} \pmod 2$

而這題有限制 $det(A) \equiv 1 \pmod 2$，所以 A^{-1} 存在。故我們可以在 $O(\frac{n^3}{64})$ 的時間內求出反矩陣並且解決這題。

4 線性基

線性基是一個用來快速解決許多子集異或和一類題目的算法。我們來看一下線性基是什麼吧。

4.1 線性基介紹

線性基其實就是一個特殊的基底 (basis)。而線性基是與異或 (xor) 相關的一個演算法，也就是說他的操作都是在 $\mathbb{Z}_2^n$ 底下進行。白話一點的說，將一個 n 位的二進制整數視爲一個由 n 個向量線性組合成的向量。這邊給出線性基的定義。

Definition. 一個集合 B 被稱爲集合 S 的線性基若且爲若滿足：

1. $S \subseteq span(B)$，即 S 可以被 B 生成
2. 集合 B 內元素線性獨立

另外定義線性基 B 的大小爲 B 的元素個數。

線性基有以下基本性質：

1. B 的任何子集都不會是線性基
2. S 中的任意一個元素都可以被唯一表示爲 B 中若干個元素 xor 起來的結果

事實上上面的這些事都是 basis 的基本定義而已，接下來我們說明如何快速的建構出線性基。

4.2 建構

令集合 S 中最大的值在二進位下有 L 位，我們使用一個陣列 $a[0 \dots L]$ 來存放線性基。

線性基是動態構造的，也就是說我們要從 a 爲空開始依序加入數字。令插入的數字爲 p，我們從 p 的最高位依序往低位掃描，若第 x 位是 1，如果 $a_x = 0$，那麼讓 $a_x = p$ 並且結束掃描。反之如果 $a_x \neq 0$，那麼令 $p = p \oplus a_x$ 繼續進行掃描。這裡給出參考程式碼

```
struct LinearBasis {
    static const int MAXL = 60;
    int a[MAXL + 1], sz = 0;
    inline void addVector(int p) {
        for(int i = MAXL; i >= 0; i --) {
            if(!(p >> i)) continue;
            if(!a[i]) {
                a[i] = p;
                sz ++;
                break;
            }
            p = p ^ a[i];
        }
    }
};
```

這個做法其實就只是在改造高斯消元而已。原因是從高位到低位依序掃過去的過程中，如果高位已經存在線性基，那麼掃過去的過程就會被消除掉。如果沒有的話直接插入，這也就是高斯消元在做的事情。由上述做法做出來的結果會形成一個上三角矩陣 (最高位當第一列)。

舉個例子，現在有三個二進制整數：$\{011, 110, 100\}$，我們依序將這三個數插進線性基的結果如下。

第幾次	a_2	a_1	a_0
1	0	011	0
2	110	011	0
3	110	011	001

當我們把三次插入都做完之後，如果我們把 a_2, a_1, a_0 依序由上而下的寫成三列的話，就會長這樣：

$$\begin{matrix} 110 \\ 011 \\ 001 \end{matrix}$$

而這也就是高斯消元後出來的結果 (上三角矩陣)。那麼複雜度呢？由上面的程式碼就可以看出每次插入都是 $O(logN)$ 的複雜度，所以整體複雜度是 $O(NlogN)$。

需要注意的是，在實用上我們通常會需要把它消成對角矩陣，所以使用上需要先處理成對角矩陣。

4.3 應用

線性基最有名的應用就在於子集最大異或和問題。

Example 4.1. 子集最大異或和問題
給定 n 個數字 a_i，選出一個子序列，並且計算他們的 xor 值，請問最大的 xor 值可以是多少。

在還沒有學線性基前，這題可能比較常見的做法是在 trie 上做 greedy。然而我們現在給出一個線性基的做法。直接上結論：求出線性基中的所有數字 xor 起來就是答案了。

證明如下：由高位到低位考慮在線性基中存在的第 j 位。如果第 j 位出現在線性基裡面，那麼整個線性基也就只有唯一一個元素的第 j 位唯一 (如果消成對角矩陣)。所以選他答案一定更優。故可以推得線性基裡面的數字必須被全選。

另外，除了上面這個問題之外，線性基也很常跟生成集一起出現。讓我們來看一個例子：

Example 4.2. CSAcademy Xor Closure
給你 n 個數字 a_i。有一個集合它包含這 n 個數字，且在這個集合中任選兩數的 xor 值仍然落在這個集合中。請問這個集合大小。

因爲線性基 (基底) 有一個性質：線性基 B 的最大生成集大小 (也就是 $span(B)$) 爲 $2^{|B|}$，所以求出線性基之後答案就會是 $2^{|B|} - n$。

線性基的題目通常都與子集異或和有關，主要有以下這幾種：

1. 最大/第 k 大異或和
2. 求所有異或值的和
3. 基底/生成集類型題目

5 矩陣樹定理

矩陣樹定理是一個處理生成樹計數問題的一個定理。由於這個定理的證明會用到特徵值跟特徵向量，故這邊只說明定理內容，不說明證明 (下面有證明連結)。

5.1 無向圖生成樹計數

給定一個圖 G，其鄰接矩陣爲 A。接下來來做一些定義。

Definition. 度數矩陣
定義 D 爲度數矩陣，其中 $D_{i,i} = \deg i$，其他部分爲 0。

Definition. Kirchhoff 矩陣
定義 K 爲 Kirchhoff 矩陣，其中 $K = D - A$。

最後我們定義 K' 爲 K 去掉第 i 列第 i 行 (i 是任意一個 $1 \sim n$ 的整數) 的子矩陣，而 $\det(K')$ 即爲生成樹個數。

5.2 有向圖生成樹計數

因爲是有向圖，所以有向圖的生成樹計數要先決定根結點，這裡令根節點爲 x。另外，我們定義外向樹爲有向邊是從根的方向往外，而內向樹爲有向邊的方向是指向根的。

Definition. 入/出度矩陣
定義 D_{in} 爲入度矩陣，其中 $D_{in\ i,i} = deg_{in}(i)$。類似定義出度矩陣 D_{out}。

Definition. 入/出度 Kirchhoff 矩陣
類似上面定義，$K_{in} = D_{in} - A$,$K_{out} = D_{out} - A$。並且定義 K'_{in} 爲刪掉第 x 列跟第 x 行的子矩陣。類似定義 K'_{out}。

此時外向樹的生成樹個數爲 $\det(K'_{in})$。內向樹的生成樹個數爲 $\det(K'_{out})$。

5.3 帶權生成樹計算

容易知道該作法是可以處理重邊的，而若我們將重邊想像成權值的話，那麼矩陣樹定理求的就是所有生成樹邊權乘積的和。所以如果要處理帶權的話就反向將他視爲重邊來進行計算即可。

5.4 整理

矩陣樹定理事實上是一個不太常出現的定理。主要是因爲他不太好包裝，而單純考這個定理又有些單調。但我認爲這個定理的證明相當有趣，礙於篇幅限制這裡不再說明，但這裡給出一個我覺得相當完整的證明：線代啓示錄 克希荷夫矩陣樹定理。

6 參考文獻

1. 2021 IOIC 講義
2. OI-WIKI 矩陣
3. 線代啓示錄 Strassen 演算法 分治矩陣乘法
4. SSerxhs 的博客 題解 P7112 【【模板】行列式求值】
5. 維基百科 餘因子矩陣
6. Menci's Blog 線性基學習筆記
7. Sengxian's Blog 線性基學習筆記
8. command_block 的博客 矩陣樹定理 (+ 行列式)