

## 演算法 PA2 (Finding MPSC)

B09901080 電機三 吳宣逸

### ■ Data Structure

PA2 中我從 input file 讀入資料後，根據行數以及第一行給定的  $n$  值，將  $n$  以及弦的數量以下方兩個變數表示：

```
int n;          // 1 <= n <= 90000
int chord_num; // number of chords
```

為了取用資料方便，我自訂了資料類型 `chord`，用  $(i, j)$  表示一條弦的兩個端點，並且不限制  $i$  和  $j$  的大小關係：

```
typedef struct {
    int i; // first pivot
    int j; // second pivot
} chord;
```

接著定義主要實現尋找 MPSC 的 4 個資料結構，它們都是 1D 或 2D dynamic array：

```
chord *chords; // array of chords use first pivot as index
int **M;       // M array of MPSC problem
char **Cases;  // Memorize cases in iterations to trace back for chords (1,2,3)
chord *varychords;
```

各變數的意義以及設計想法如下：

#### 1. chord \*chords

`chords` 是大小  $2n$  的 dynamic 1D array，將第一個端點是  $i$  的弦存為 `chords[i]`，例如  $(8, 11)$  會被存在 `chords[8]`。另外由於弦的數量  $\leq n$ ，因此若第  $i$  個點不是任何弦的端點，則用 `chords[i] = (-1, -1)` 作為不是弦的標記。

為了能快速取用，一條弦除了存成  $ij$  之外還另外存一份  $ji$ ，儘管用掉兩倍空間，但可將找弦  $jk$  是否存在的時間複雜度由  $\Omega(\lg n)$  (將弦存成 `Balanced Binary Tree` 再 `search` 應該是用最少空間最快的做法) 變成  $O(1)$ ，因為取用 `chords` 的次數很多，所以會大大減少 `search` 所需時間。

#### 2. int \*\*M

$M$  是大小  $(2n-1) \times m(i)$  的 dynamic 2D array， $M[i'][j']$  儲存弦  $ij$  的 MPSC 數量 ( $i' = i, j' = j - i$ ，原因之後說明)，其中

$$m(i) = \begin{cases} 2n - i, & M[i'] \text{ used} \\ 0, & M[i'] \text{ not used} \end{cases}$$

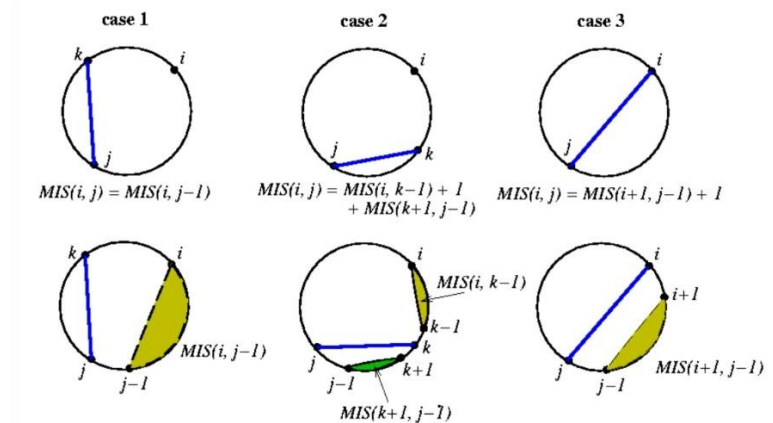
，因此  $M[i']$  皆初始成 `nullptr`，若是  $M[i']$  須要儲存資料才會被 `new` 成新的陣列。 $M$  的所有值都是預設 `-1`，代表空值；而 `base case` 是當  $i = j$  時  $M[i'][j'] = 0$ 。

另外，由於實際上不論如何第  $i$  行只有後  $(2n - i)$  個 `element` 會用於儲存資料 ( $i \leq j$ )，因此  $m(i)$  的大小不需要到  $2n$  而是  $(2n - i)$  即足夠，也因為是後  $(2n - i)$  個 `element`，因此才有前面提到的  $j' = j - i$  的 `mapping`。

M 之所以每行在需要時才做動態配置是因為 M 會有好幾行完全不被用到，因此才做事後動態配置以節省大量的時間、空間。

### 3. char \*\*Cases

除了 Cases[i][j] 儲存的是 M[i][j] 對應的 case 如下，以及為了節省空間以 char 儲存之外，其餘操作方式和設計想法與 M 完全相同。Cases 所有值都預設 '0'，代表空值。



### 4. chord \*varychords

用於儲存 MPSC 包含的所有弦，操作方式和設計想法與 chords 完全相同。最後會再剔除弦 ij、ji 中第一個端點較大者，並將剩餘的弦回傳，即求得自動排序好的 MPSC 了。

## Algorithm Introduction

PA2 我使用的是 top-down with memorization 的作法，主要是實做過 bottom-up 之後發現會進行許多不必要的運算，並且使用空間也較大，雖然聽說有同學使用 bottom-up 的方法也能跑出不錯的時間和空間，但可能 PA2 使用 top-down with memorization 在設計上仍是比較簡單。

下方將會簡述演算法的想法，在此之前，各 case 對應的遞迴式如下：

	Case 1	Case 2	Case 3
Number of MPSC	$MIS(i, j) = MIS(i, j-1)$	$MIS(i, j) = MIS(i, k-1) + 1 + MIS(k+1, j-1)$	$MIS(i, j) = MIS(i+1, j-1) + 1$
MPSC	Do nothing	Add chord (j, k) to varychords, then go (i, k-1) and (k+1, j-1).	Add chord (i, j) to varychords, then go (i+1, j-1).

整體演算法可以分成三個步驟：

1. 根據 (i, j) 以及弦 jk 判斷對應的 case 並記錄到 Cases[i][j]，再根據 case 對應的遞迴式填入 M[i][j]。實際做法是從 (i, j) = (0, 2n-1) 遞迴，結束後 M、Cases 所需值就被填好了。
2. 回傳 MPSC 數量，也就是 (i, j) = (0, 2n-1) 的 M[i][j] 值。
3. 根據 Cases 回溯 MPSC 包含的所有弦，每次將對應的弦加入 varychords。

## My Finding

在過去寫 DP 的 pseudo code 時，常常忽略節省空間的重要性，使得在 HW2 第 10 題 MPSC 問題

也是一次就切出  $2n \times 2n$  的  $M$  出來，並且使用想法較簡單的 **bottom-up** 方法，然而在這次實作 **MPSC** 之後才發現原來可以在時間、空間上做到這麼大量的優化，能夠讓原本要跑 16 分鐘的程式變成在不到 1 秒就能完成。

其實我最一開始認為動態配置非常花時間，因此想全部配置成 **static array** 但失敗了，造成前面所述的 16 分鐘。之後改成 **dynamic array** 後發現效能沒預期中差，才想到 **dynamic array** 最耗時的只有在 **allocate memory** 而已，由於一次不用切出這麼多空間，而且每行最多也只要 **allocate** 一次即可，因此遠比開一個 **static array** 省時、省空間許多。這再次提醒我 **time** 和 **space** 不是永遠都是 **tradeoff**，有時兩者是密切相關的，因此應該要先看清兩者的關係才能夠設計出好的演算法。